Árvore B

Prof. Bárbara Quintela
Prof. Jose J. Camata
Prof. Marcelo Caniato
barbara@ice.ufjf.br
camata@ice.ufjf.br
marcelo.caniato@ice.ufjf.br

Tópicos

1. Introdução
2. Propriedades
3. Inserção
4. Remoção
5. Variações

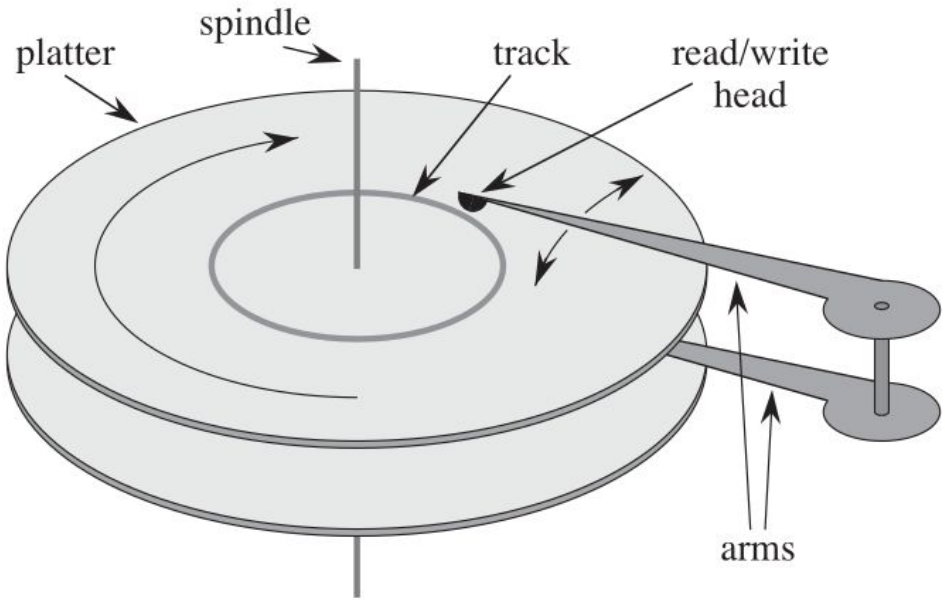
Introdução

- Se temos um grande volume de dados, pode não ser possível mantê-los na memória principal
  - Necessidade de trabalhar com **memória secundária**
  - **Problemas**
    - O custo de acesso ao disco é muito maior
      - Da ordem de **milissegundos**, ao passo que na memória principal é da ordem de **nanossegundos**
    - **Árvores binárias de busca não são eficientes em disco**

Introdução

- Se temos um grande volume de dados, pode não ser possível mantê-los na memória principal
  - Necessidade de trabalhar com **memória secundária**
  - **Problemas**
    - O custo de acesso ao disco é muito maior
      - Da ordem de **milissegundos**, ao passo que na memória principal é da ordem de **nanossegundos**

Introdução



FONTE: [CORMEN, 2009]

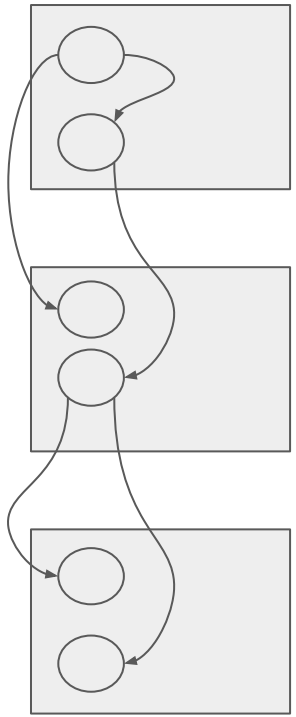
- Leitura e escrita em disco é feita por **blocos**
  - Tamanho do bloco varia de sistema para sistema
- Custo de acesso influenciado por:
  - Tempo de busca (seek)
  - Atraso rotacional
  - Transferência
- Para um disco de 7200RPM
  - Uma rotação = 8.33ms
  - Acesso à memória principal = aproximadamente 50ns
  - **Memória é acessada 100 mil vezes durante uma rotação!**

Introdução

Para manter uma árvore binária de busca no disco encontramos algumas limitações:

localidade - como elementos são adicionados em ordem aleatória, não tem como garantir que um nó recém criado esteja escrito próximo a seu pai

- ponteiros para filho podem envolver várias páginas de disco



Introdução

Uma implementação simples de ABB poderia realizar tantas buscas (acessos) no disco quanto comparações

- por não ter o conceito de **localidade** embutida

Como minimizar o número de acessos ao disco?

Introdução

Uma implementação simples de ABB poderia realizar tantas buscas (acessos) no disco quanto comparações

- por não ter o conceito de **localidade** embutida

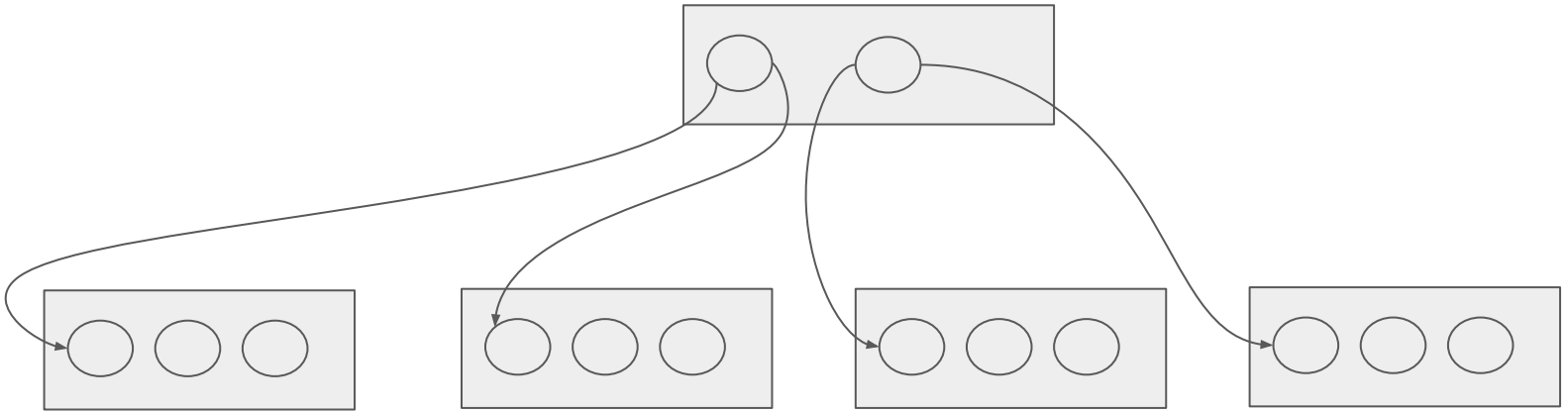
Como minimizar o número de acessos ao disco?

Ideia: colocar chaves relacionadas em locais próximos no disco

Introdução

E se tivéssemos uma árvore com um número maior de filhos?

- centenas, milhares…
- de acordo com o tamanho do bloco da memória secundária



Árvores B

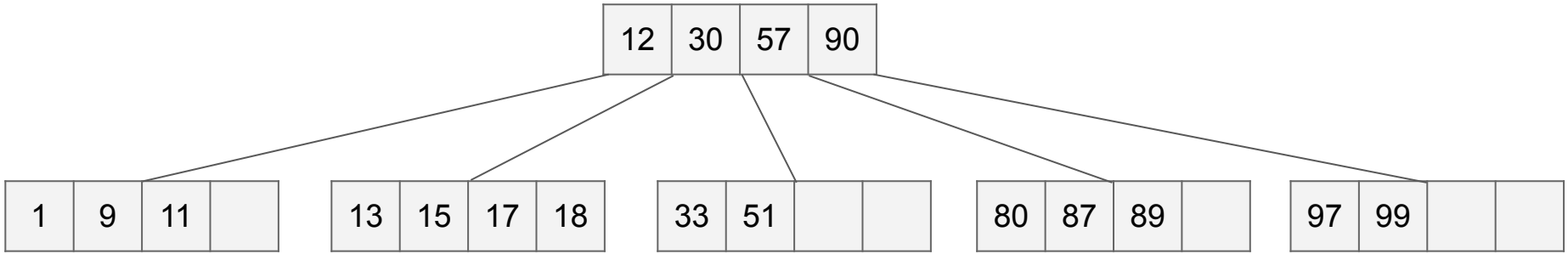
- Proposta por Bayer e McCreight em 1972
- Permite acesso eficiente a chaves em memória secundária
- Tamanho do nó definido em função do tamanho do bloco
  - Cada nó tem mais de uma chave
  - Número de chaves varia em função do seu tamanho e das partes do registro armazenadas no nó
- Perfeitamente balanceada[1]
- Construção bottom-up
- Amplamente utilizada em bancos de dados

---

1 o número de nós nas sub-árvores esquerda e direita diferem em no máximo 1

Árvore B

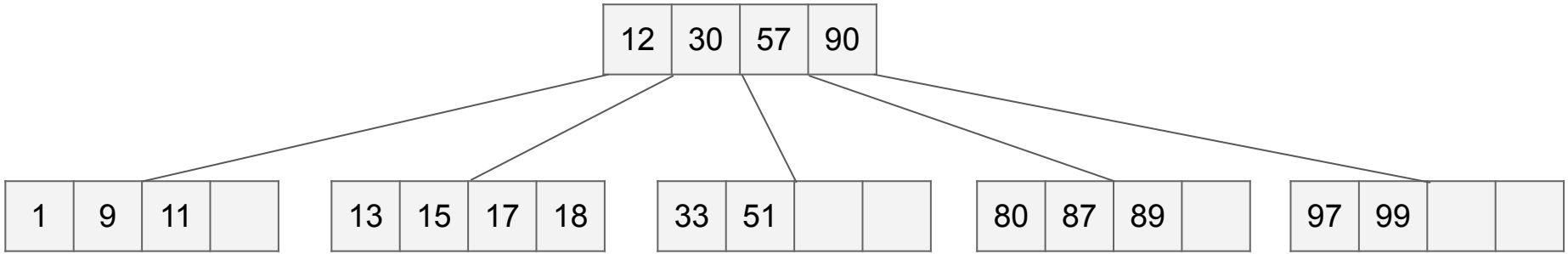
- A árvore B é um tipo de árvore conhecido como **multiway**



  - **Árvore de ordem m**: no máximo m filhos e até m-1 chaves por nó
  - O exemplo acima é uma árvore de ordem 5

Árvore B

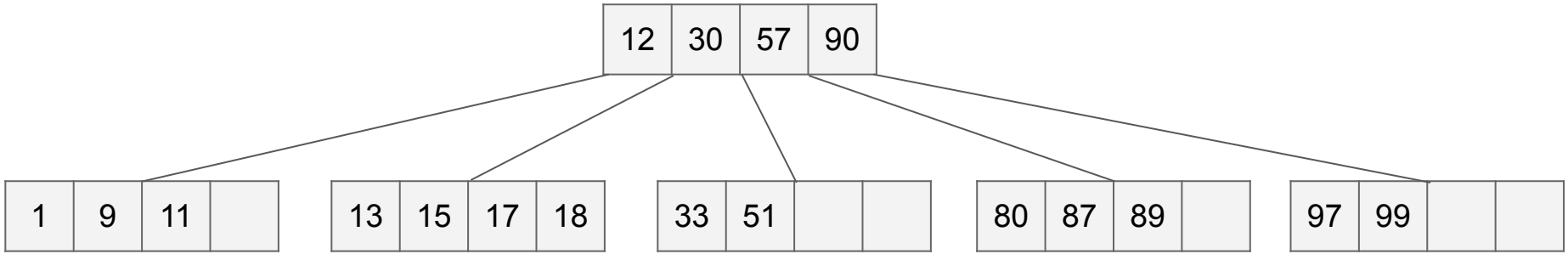
- A árvore B é também uma árvore de **busca**



  - Propriedade de ordenação similar à da ABB
    - Chaves estão em ordem crescente
    - Chaves nos filhos[0..i] < chave[i]
    - Chaves nos filhos[i+1..m] > chave[i]

Árvore B

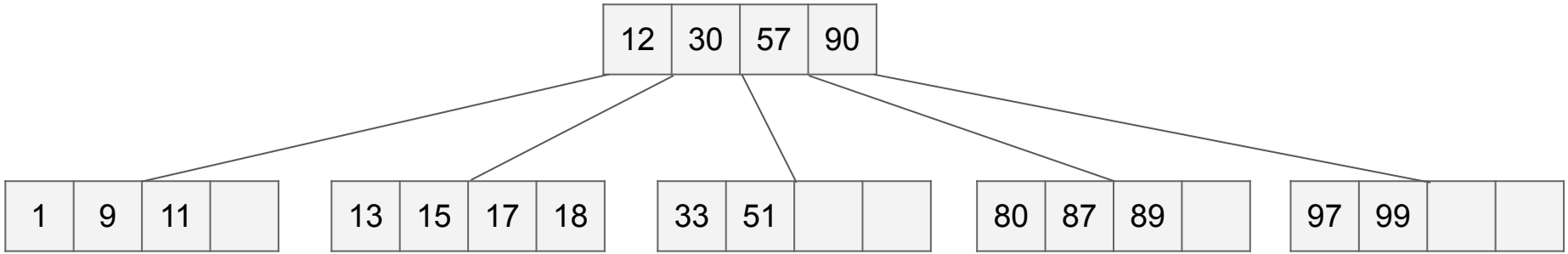
- Além disso, possui as seguintes **propriedades**



  - A raiz tem no mínimo 2 subárvores ou é folha
  - Nós internos têm k filhos e k-1 chaves, $\lceil m/2 \rceil \leq k \leq m$
  - Nós folha têm k-1 chaves, $\lceil m/2 \rceil \leq k \leq m$

Árvore B

➢ Além disso, possui as seguintes **propriedades**

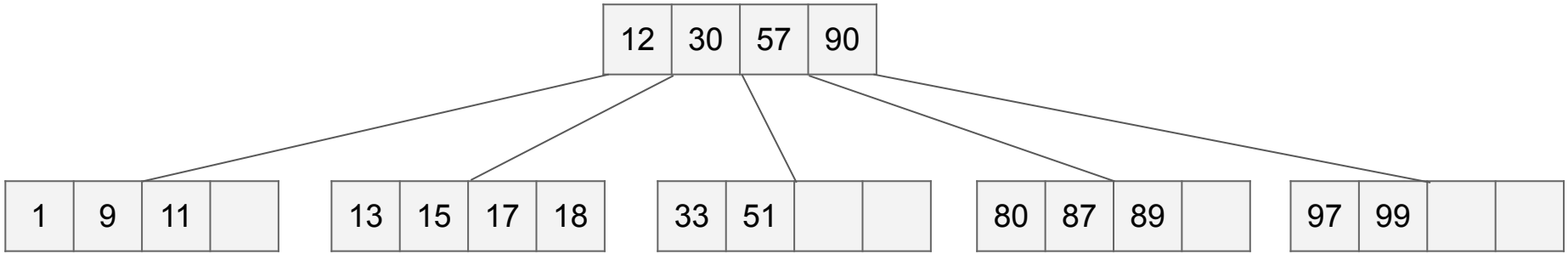


- A raiz tem no mínimo 2 subárvores ou é folha
- Nós internos têm k filhos e k-1 chaves, $\lceil m/2 \rceil \leq k \leq m$
- Nós folha têm k-1 chaves, $\lceil m/2 \rceil \leq k \leq m$

Ex.: m = 5 ⇒ Mínimo de filhos = 3 Máximo = 5
⇒ Mínimo de chaves = 2 Máximo = 4

Árvore B

- Além disso, possui as seguintes **propriedades**



  - A raiz tem no mínimo 2 subárvores ou é folha
  - Nós internos têm k filhos e k-1 chaves, $\lceil m/2 \rceil \leq k \leq m$
  - Nós folha têm k-1 chaves, $\lceil m/2 \rceil \leq k \leq m$

Obs.: Algumas definições usam o grau mínimo (número mínimo de filhos) ao invés do máximo

Árvore B

ESTRUTURA DO NÓ

```
TAD NoB
    T chaves[m-1];
    NoB* filhos[m];
    inteiro n;
    booleano folha;
```

Array com capacidade para m-1 chaves

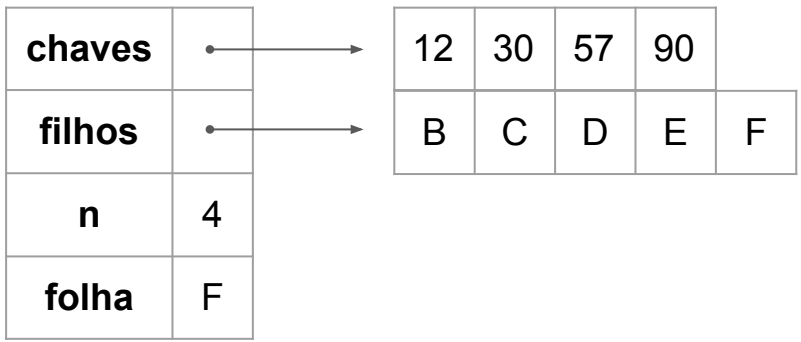
Array de ponteiros para os m filhos

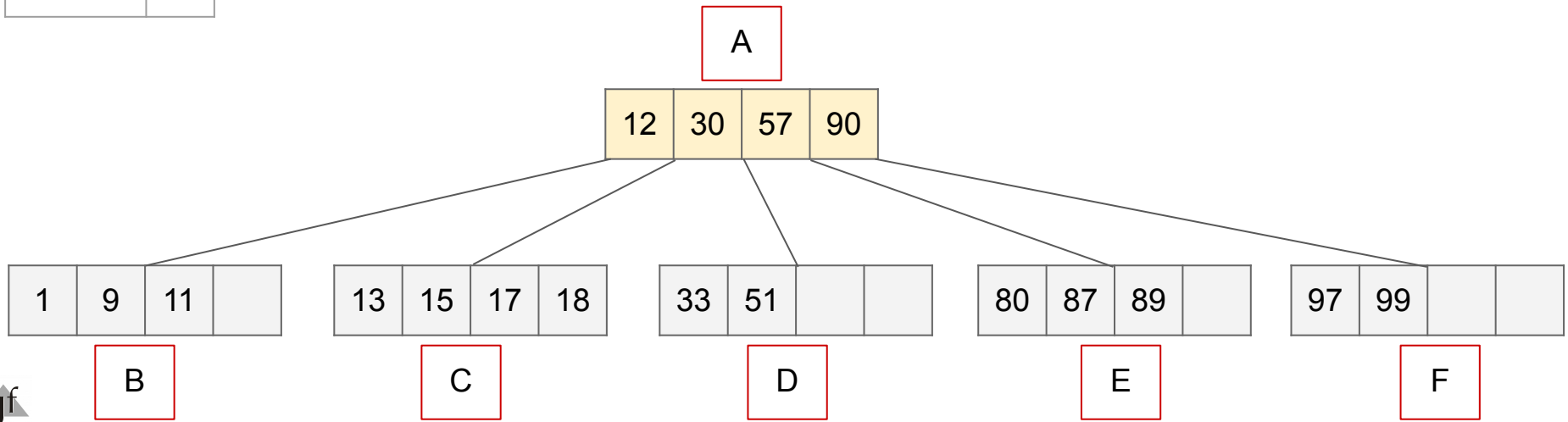
- filhos[0] é filho esquerdo de chaves[0]
- filhos[1] é filho esquerdo de chaves[1] e filho direito de chaves[0]
- …
- filhos[m-2] é filho esquerdo de chaves[m-2] e filho direito de chaves[m-3]
- filhos[m-1] é filho direito de chaves[m-2]

Indica se o nó é folha ou não

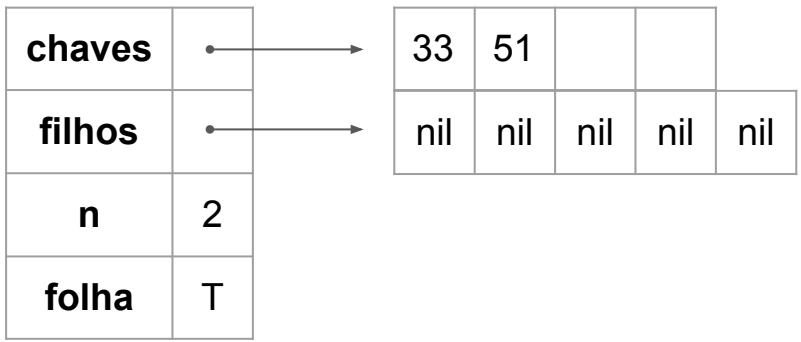
Número de chaves presentes no nó

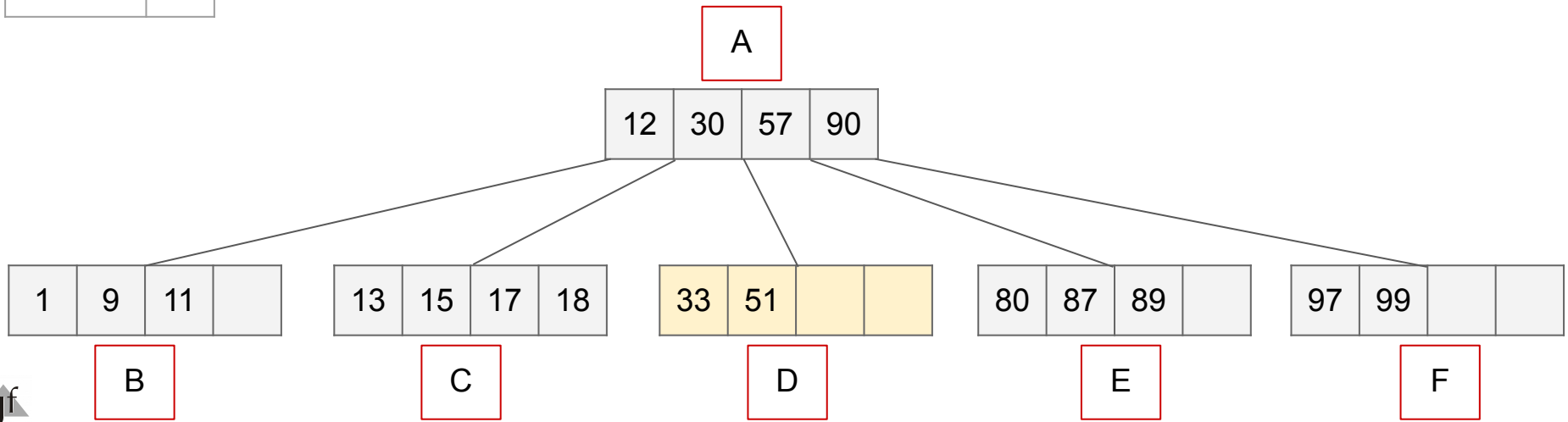
Árvore B


chaves
filhos
n
4
folha
F
12
30
57
90
B
C
D
E
F


A
12
30
57
90
1
9
11
13
15
17
18
33
51
80
87
89
97
99
B
C
D
E
F

Árvore B


chaves
filhos
n
2
folha
T
33
51
nil
nil
nil
nil
nil

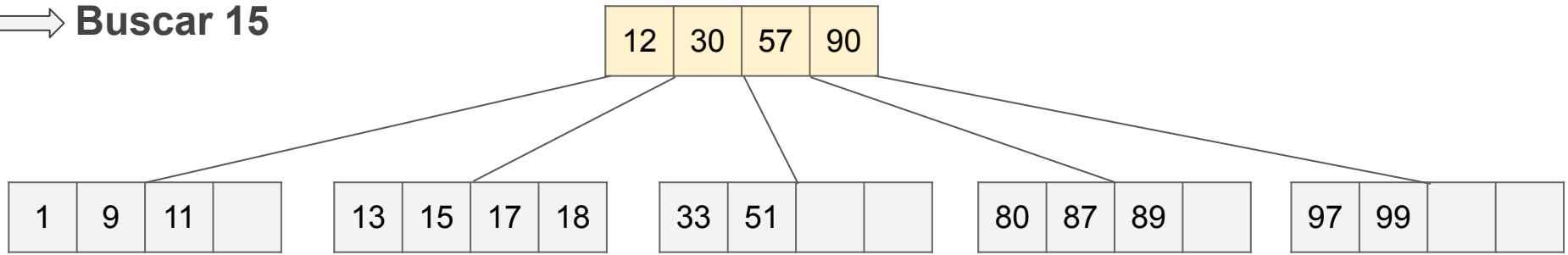

A
12
30
57
90
1
9
11
13
15
17
18
33
51
80
87
89
97
99
B
C
D
E
F

Busca

- A busca segue a mesma lógica da árvore binária de busca
- Algoritmo
  1. Inicie o procedimento pela raiz
  2. Se a árvore está vazia, retorne falso
  3. Encontre a posição no vetor de chaves onde o valor deveria se encontrar
  4. Se encontrado, retorne verdadeiro; caso contrário, desça para o nó filho à esquerda ou à direita, dependendo da ordenação em relação à chave procurada
  5. Repita os passos 3 e 4 até encontrar a chave ou atingir uma subárvore vazia

Exemplo

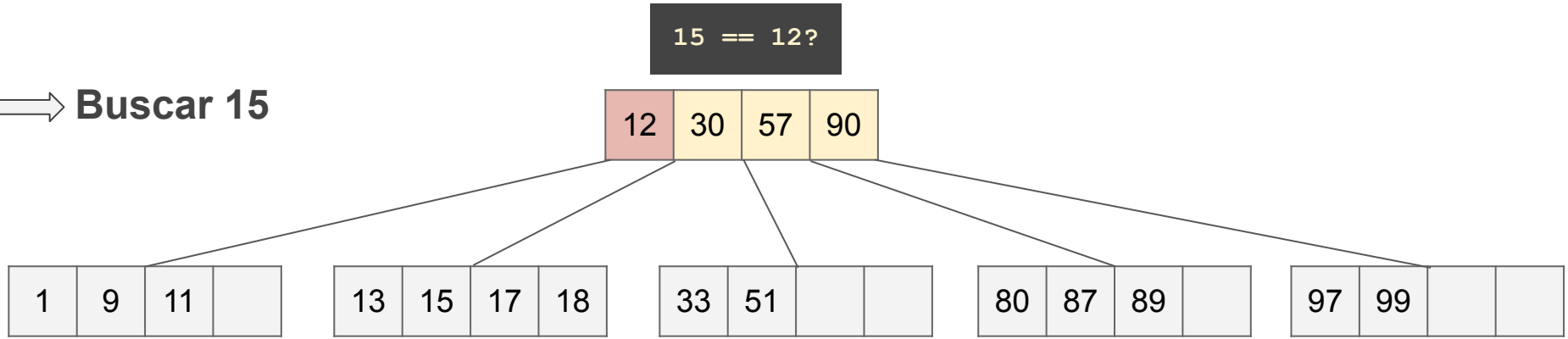
⇨ **Buscar 15**



`Inicia a busca pela raiz`

Exemplo

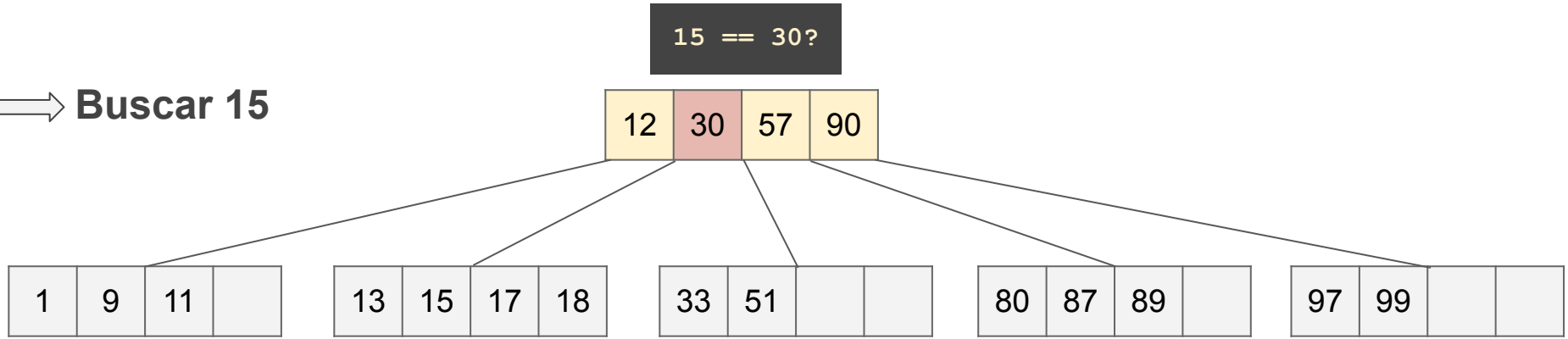
Buscar 15



Procura pela chave no nó corrente

Exemplo

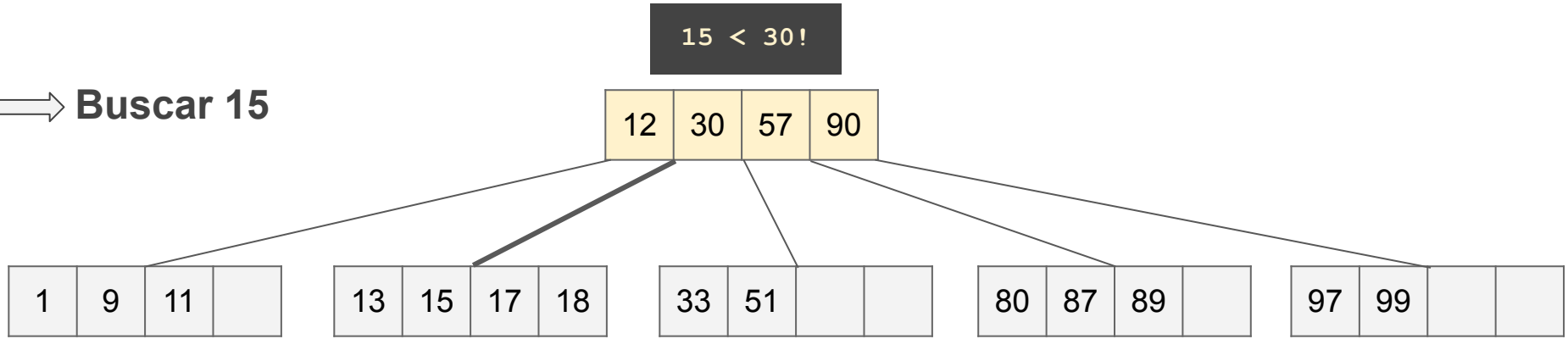
⇨ **Buscar 15**



`Procura pela chave no nó corrente`

Exemplo

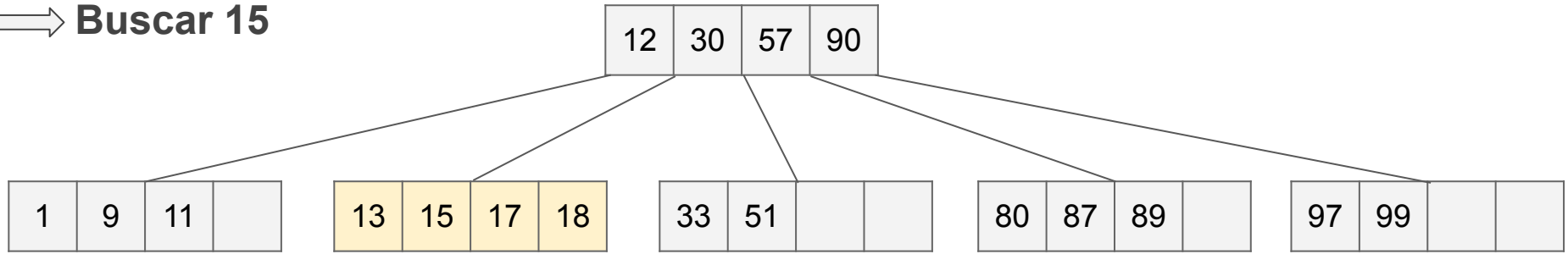
⇨ **Buscar 15**



Chave não se encontra no nó, vai para o filho

Exemplo

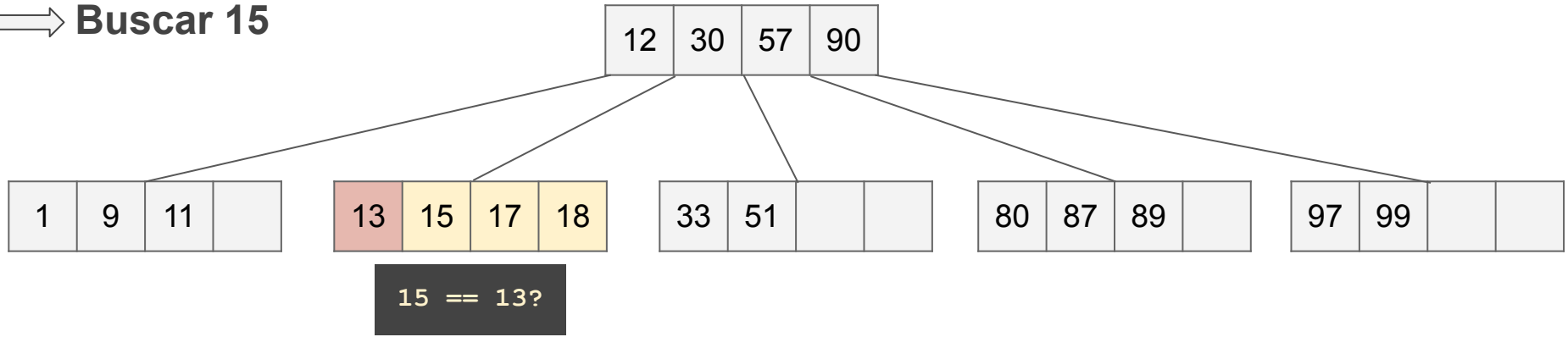
⇨ **Buscar 15**



Chave não se encontra no nó, vai para o filho

Exemplo

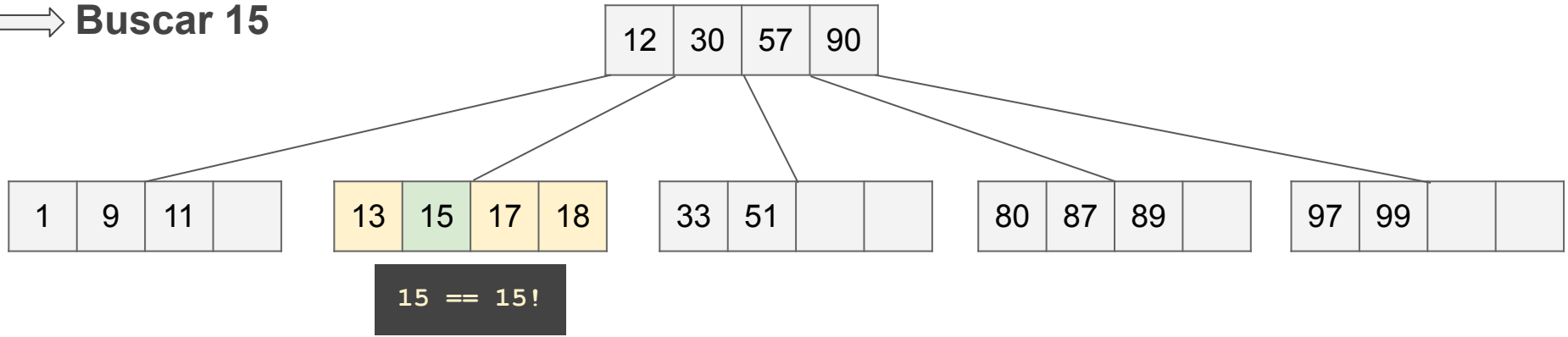
⇨ **Buscar 15**



Procura pela chave no nó corrente

Exemplo

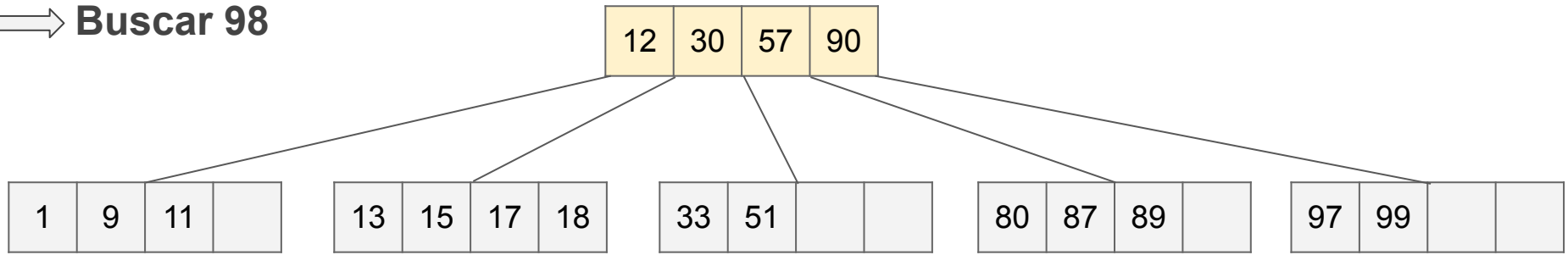
Buscar 15



Chave encontrada!

Exemplo

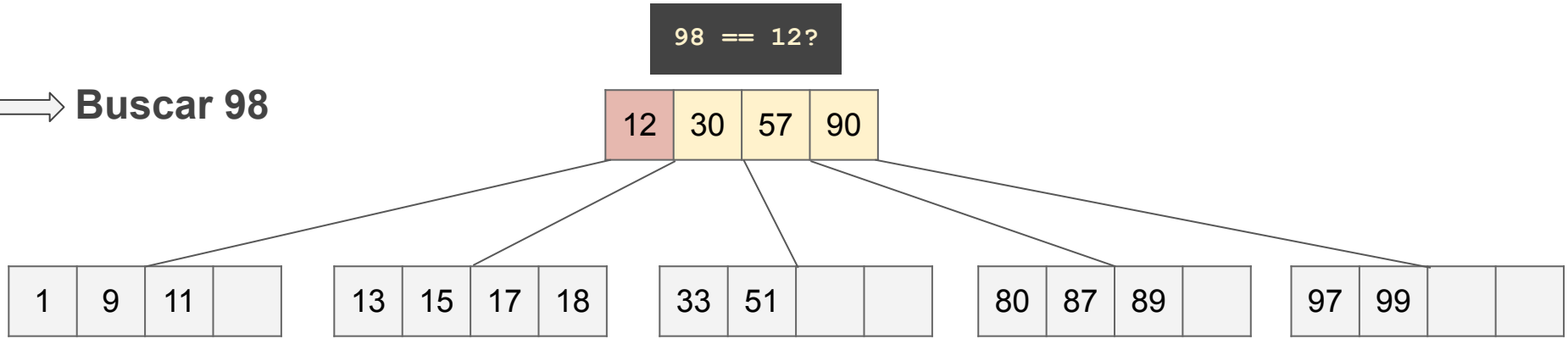
Buscar 98



Inicia a busca pela raiz

Exemplo

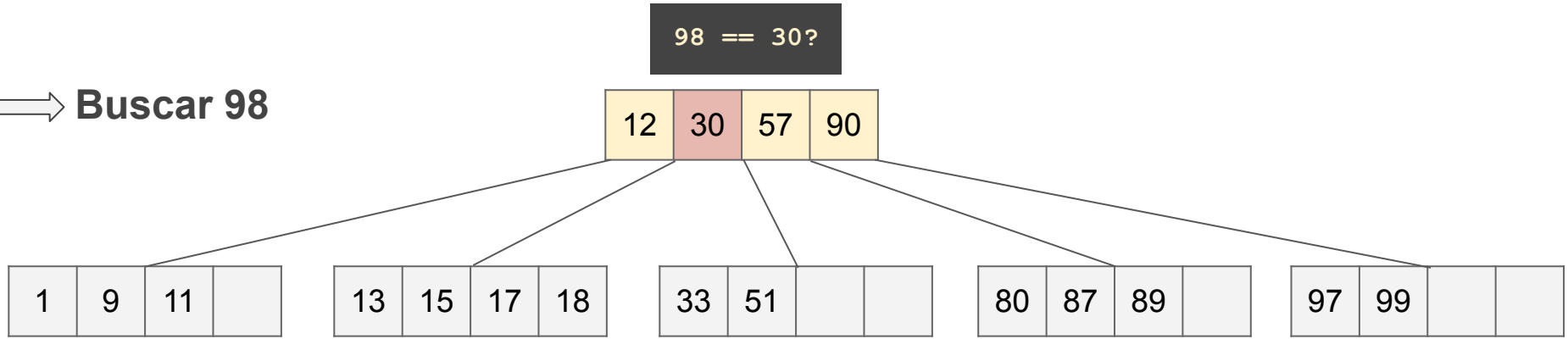
Buscar 98



Procura pela chave no nó corrente

Exemplo

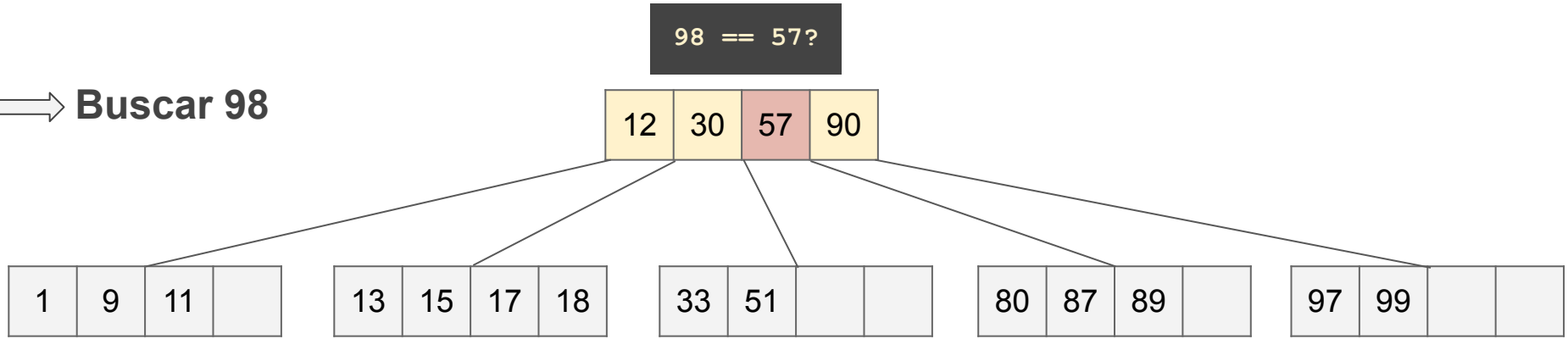
⇨ **Buscar 98**



`Procura pela chave no nó corrente`

Exemplo

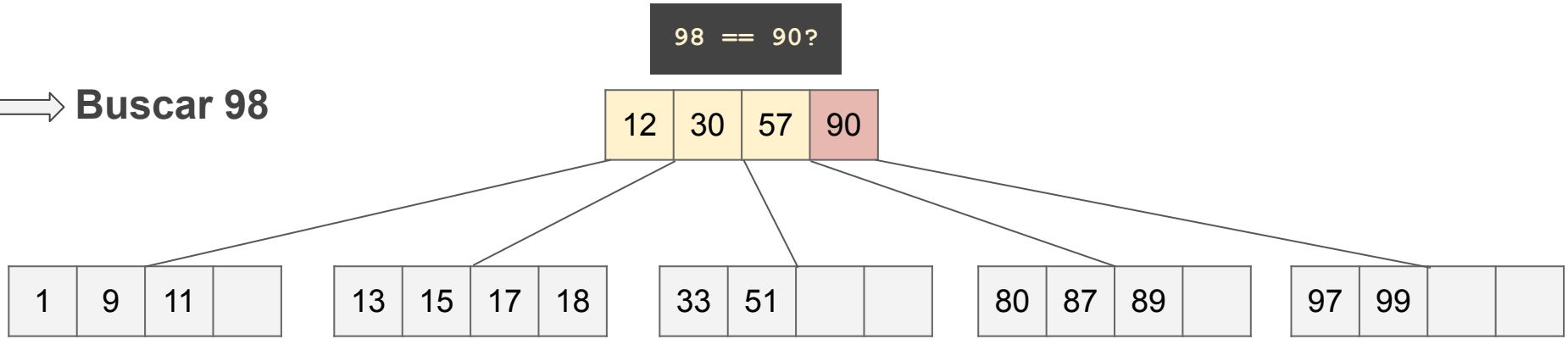
⇨ **Buscar 98**



`Procura pela chave no nó corrente`

Exemplo

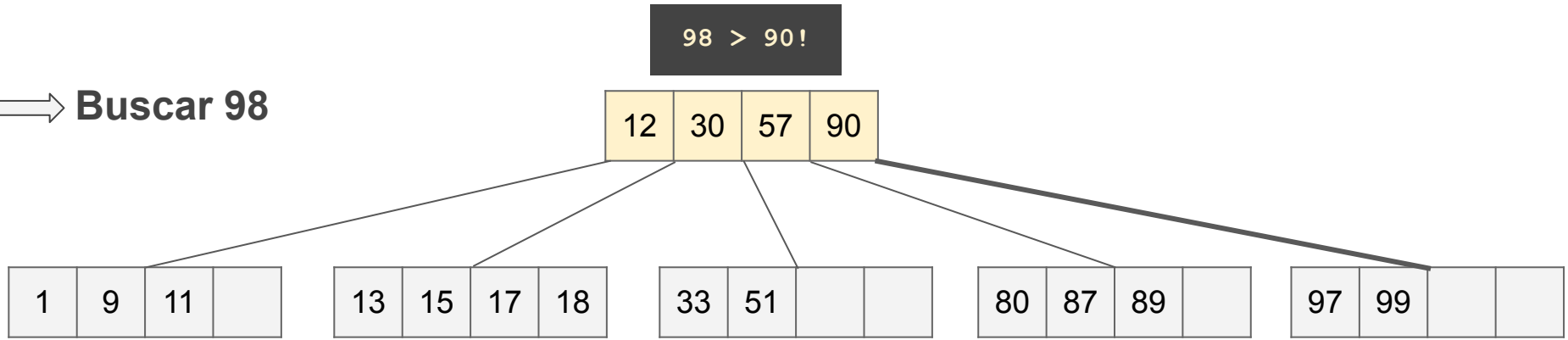
Buscar 98



Procura pela chave no nó corrente

Exemplo

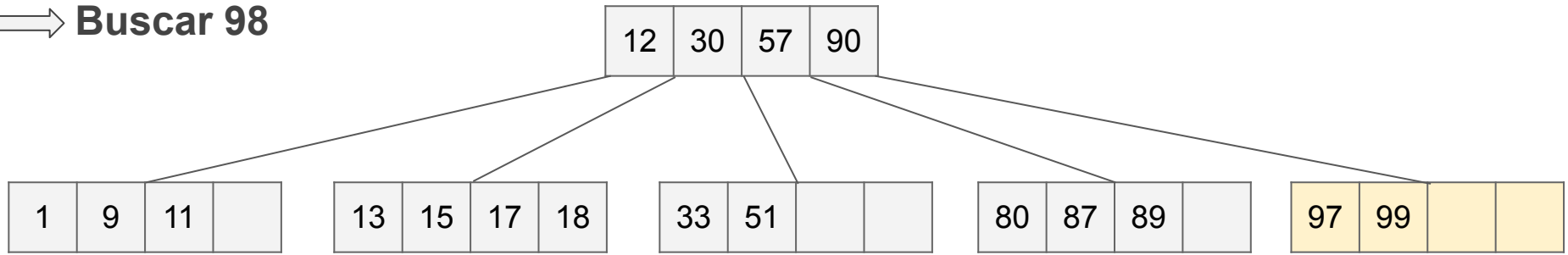
⇨ **Buscar 98**



Chave não se encontra no nó, vai para o filho

Exemplo

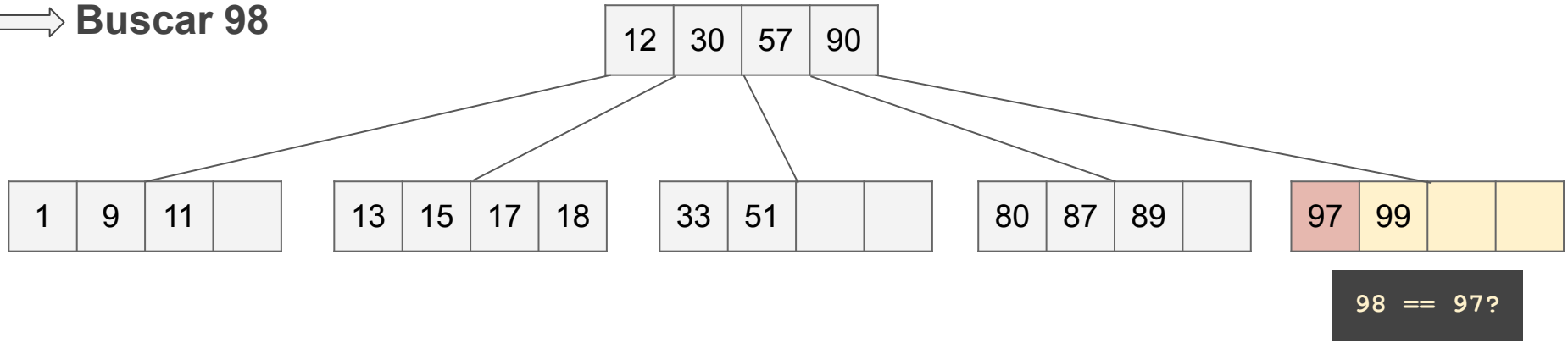
⇨ **Buscar 98**



Chave não se encontra no nó, vai para o filho

Exemplo

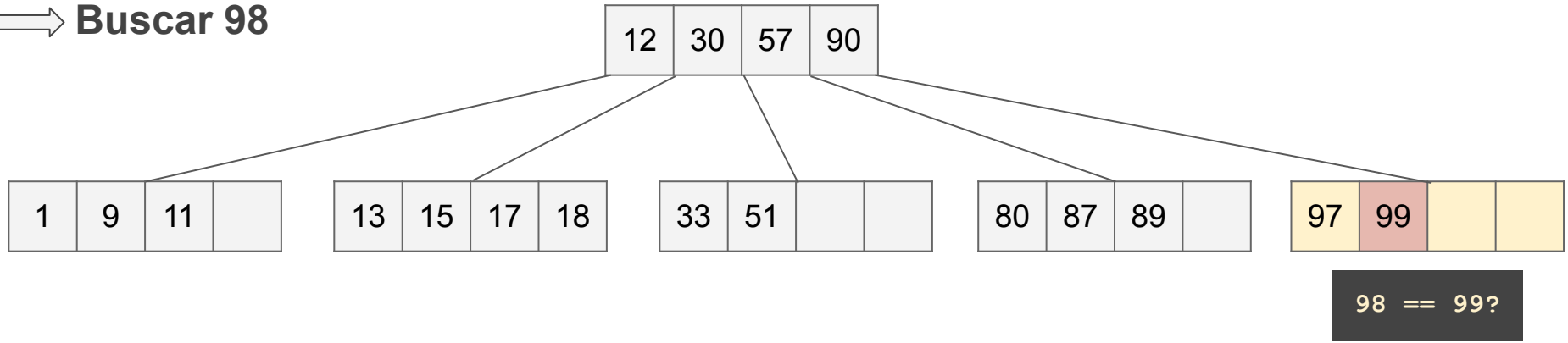
Buscar 98



Procura pela chave no nó corrente

Exemplo

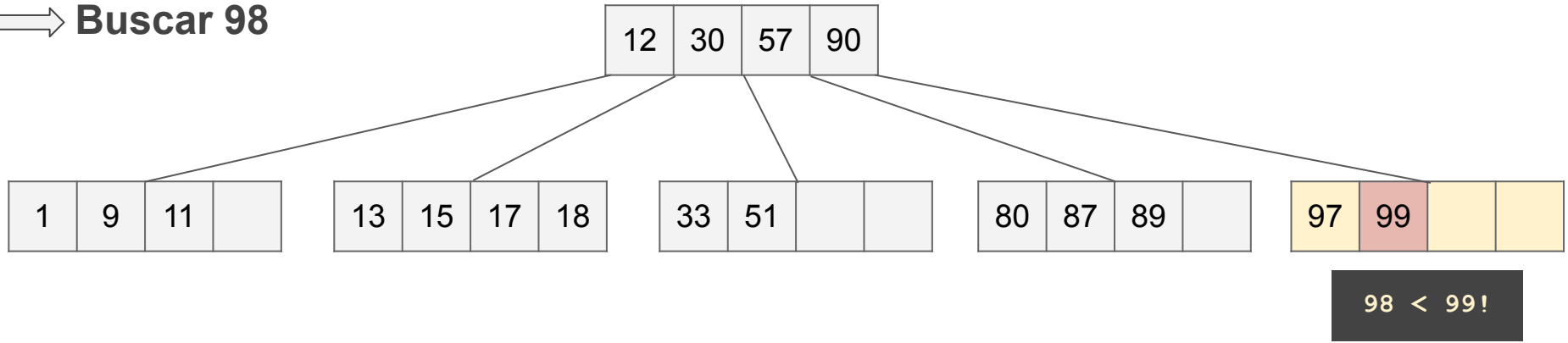
Buscar 98



Procura pela chave no nó corrente

Exemplo

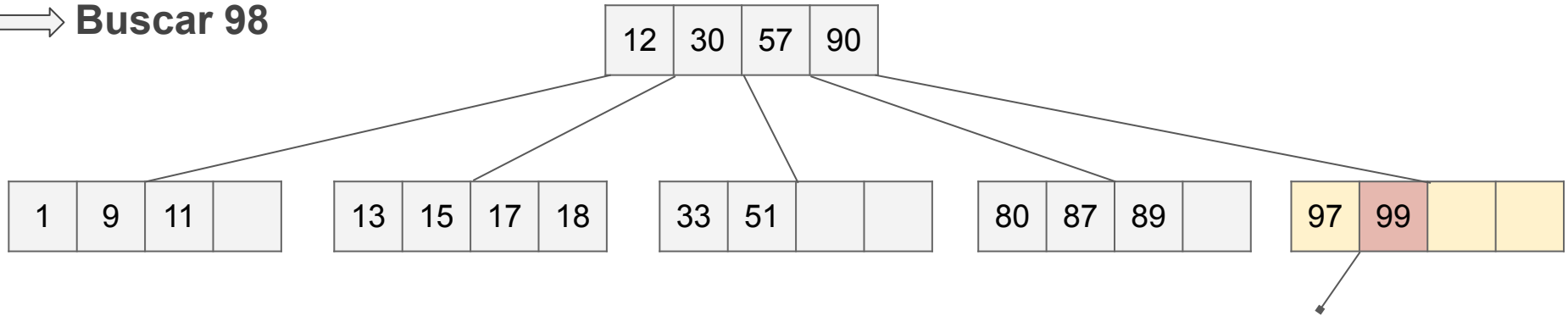
⇨ **Buscar 98**



Chave não se encontra no nó e nó é folha

Exemplo

Buscar 98



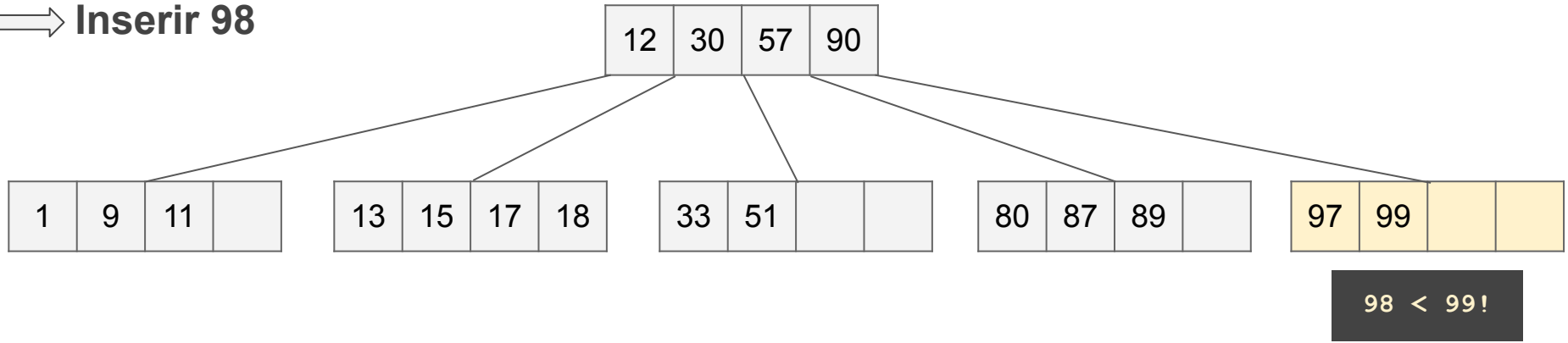
Chave não encontrada na árvore!

Inserção

- Assim como na ABB, sempre realizada em uma folha
- A nova chave é inserida preservando a ordenação do nó
- Pode ocorrer *overflow*
  - Nó já possui número máximo permitido de chaves
  - Aplicar procedimento de **cisão**
    - Um novo nó é criado, irmão do nó de inserção
    - O conjunto formado pelas chaves pré-existentes mais a nova chave é dividido entre os dois nós (antigo e novo)
    - A chave central do conjunto é promovida para a raiz, tendo como filhos esquerdo e direito os nós resultantes da cisão

Exemplo

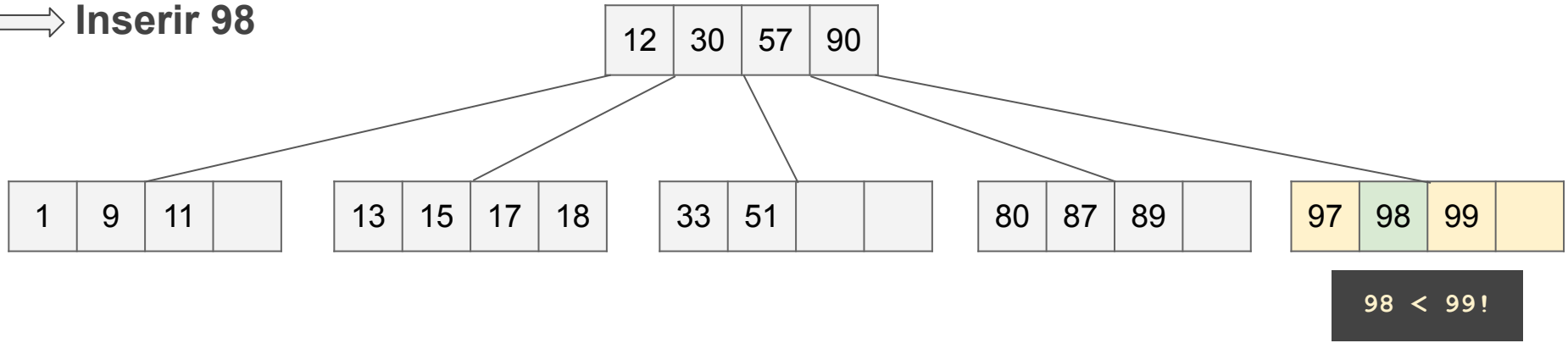
⇨ **Inserir 98**



Nó é folha, posição de inserção encontrada

Exemplo

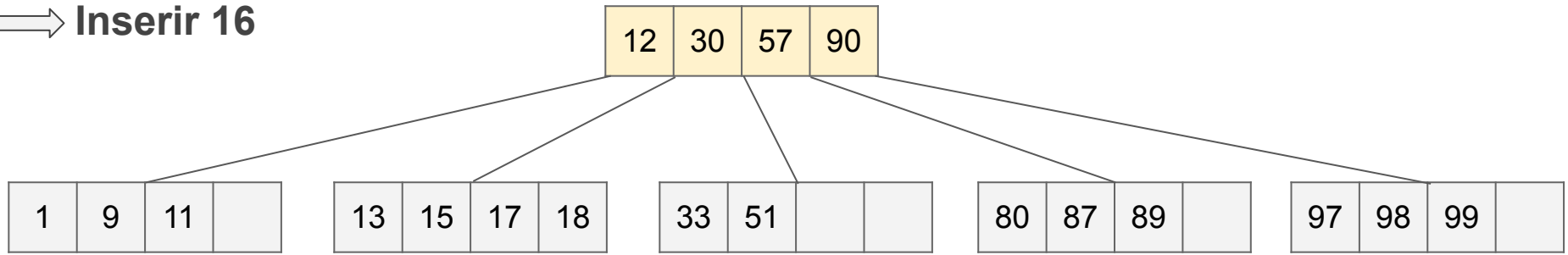
⇨ Inserir 98



Chave é inserida de forma ordenada

Exemplo

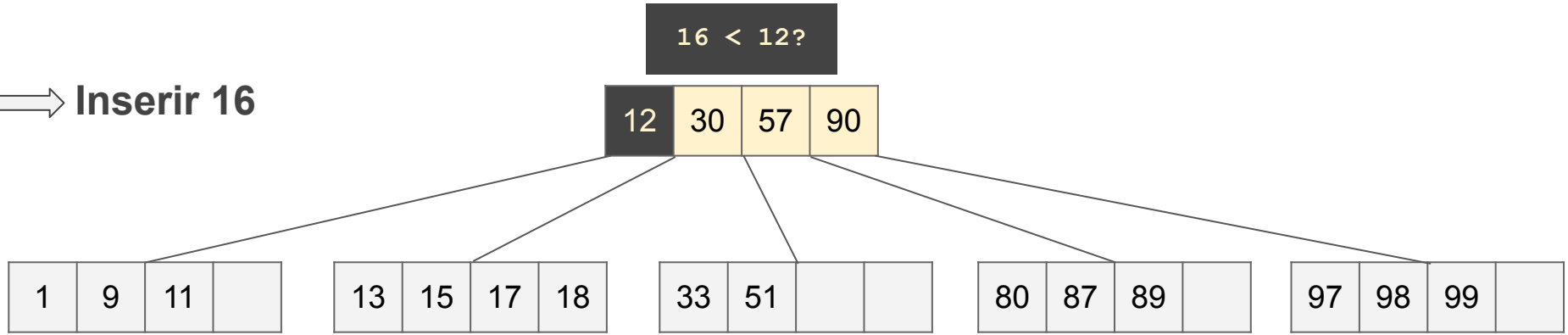
⇨ **Inserir 16**



`Inicia busca pela posição de inserção na raiz`

Exemplo

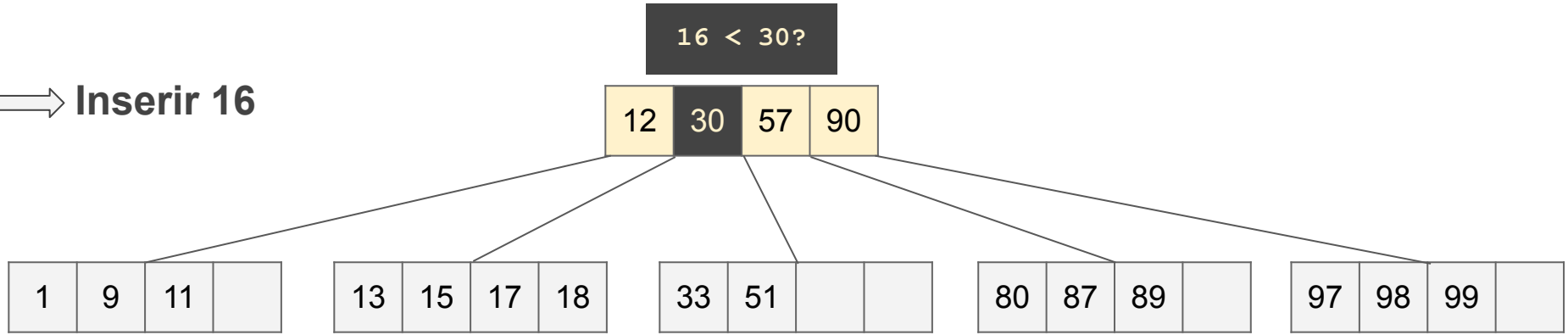
⇨ Inserir 16



Inicia busca pela posição de inserção na raiz

Exemplo

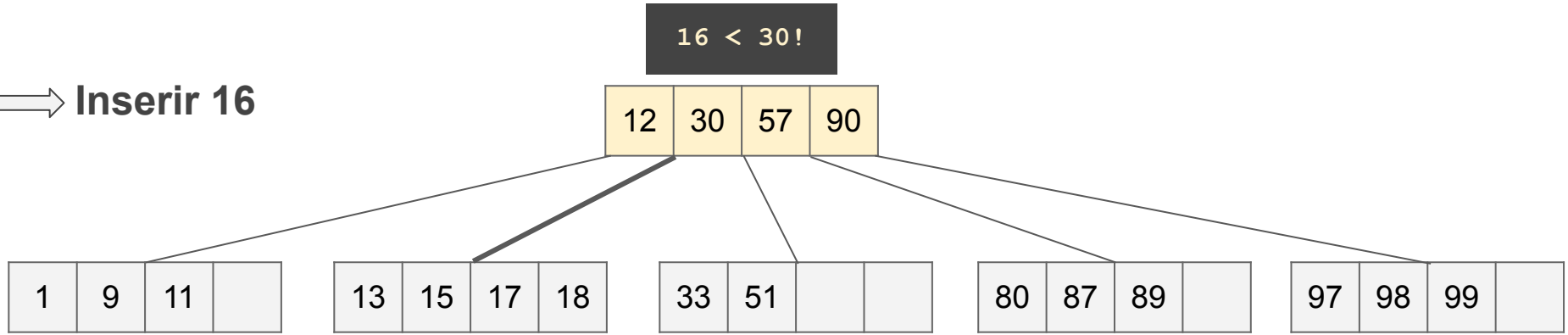
⇨ Inserir 16



Inicia busca pela posição de inserção na raiz

Exemplo

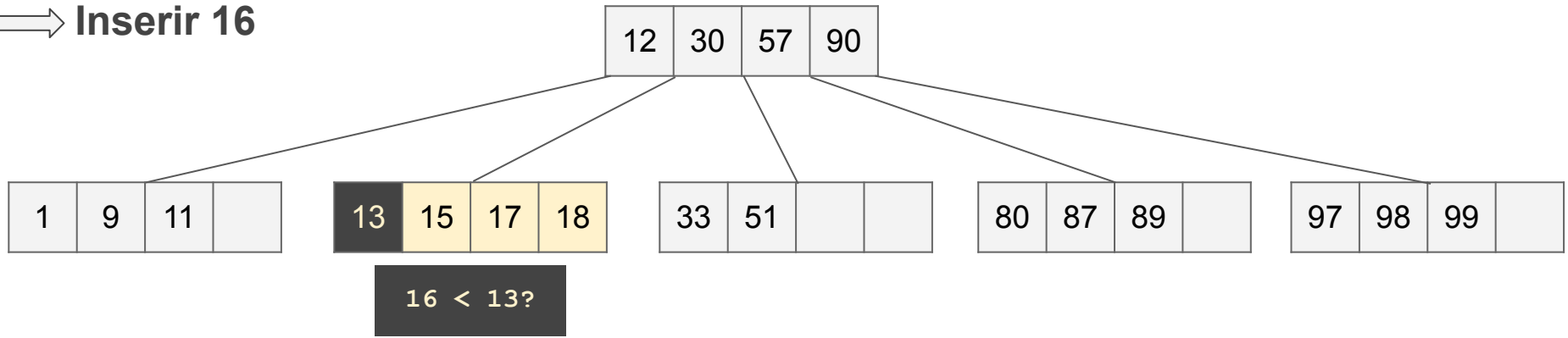
⇨ **Inserir 16**



Nó não é folha, segue para o filho

Exemplo

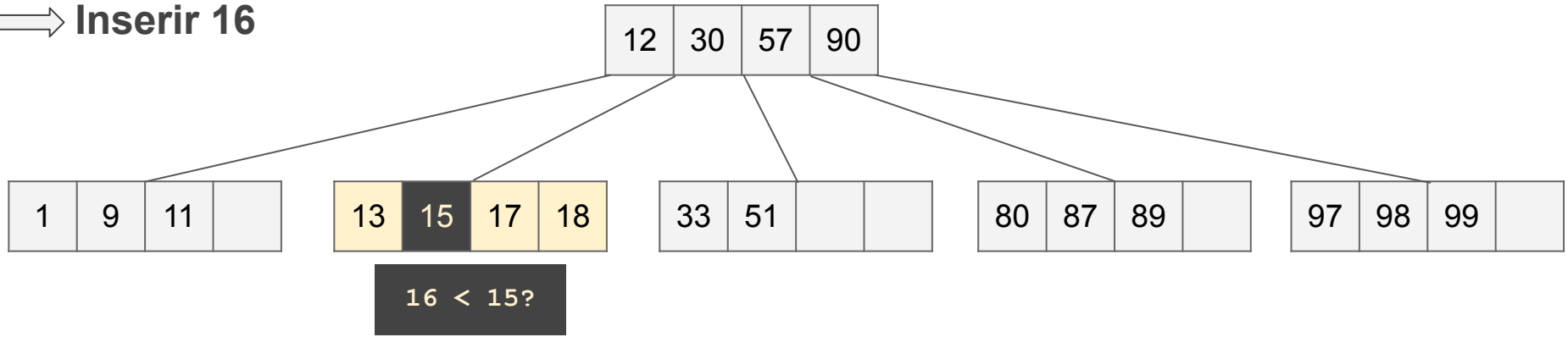
⇨ Inserir 16



Continua a busca no filho

Exemplo

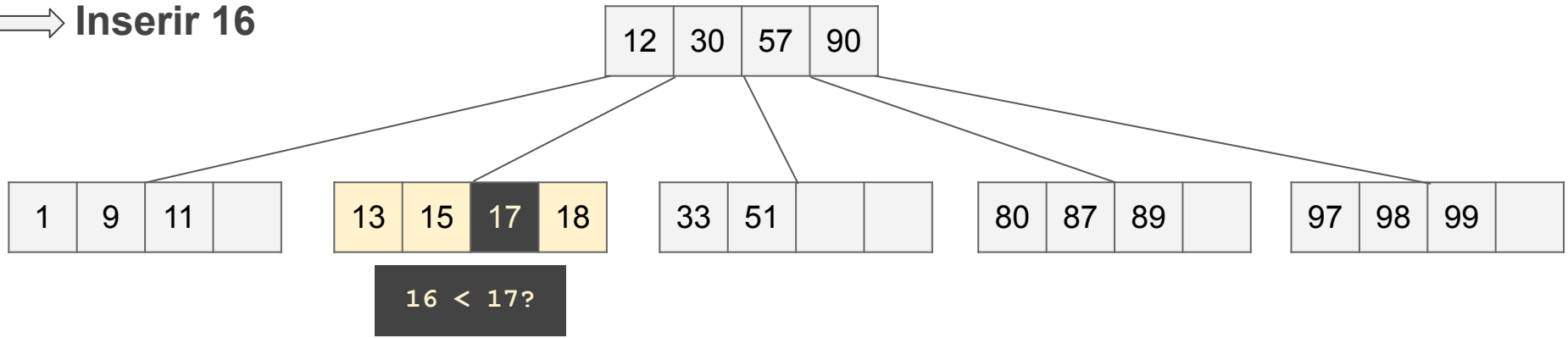
⇨ Inserir 16



Continua a busca no filho

Exemplo

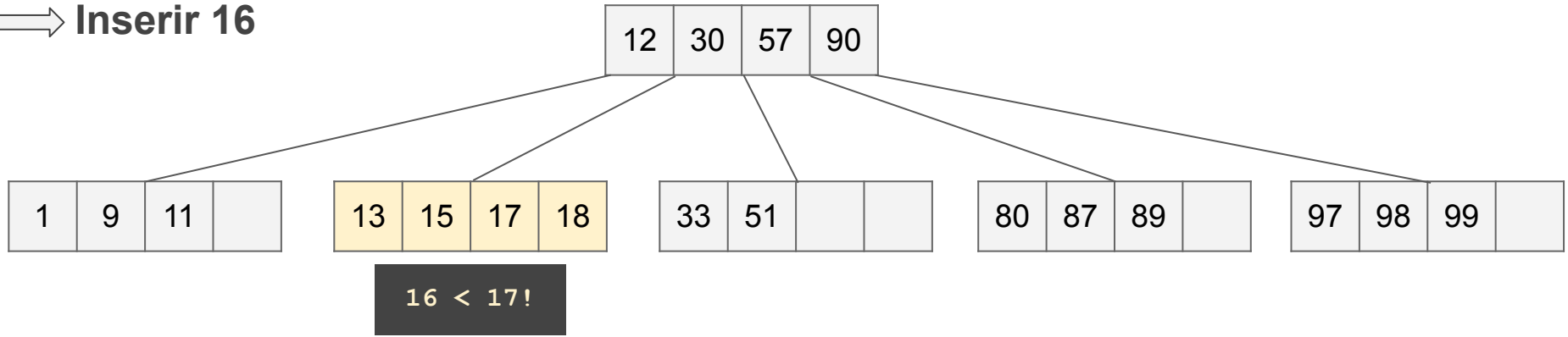
⇨ Inserir 16



Continua a busca no filho

Exemplo

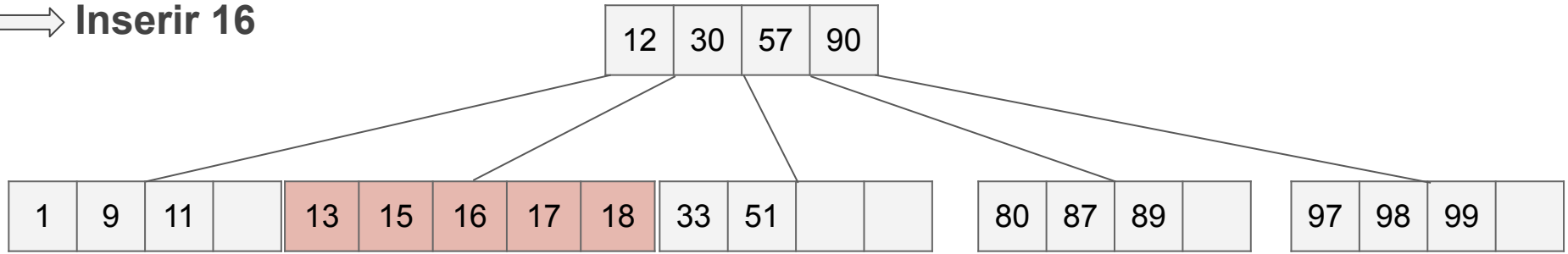
⇨ Inserir 16



Nó é folha, posição de inserção encontrada

Exemplo

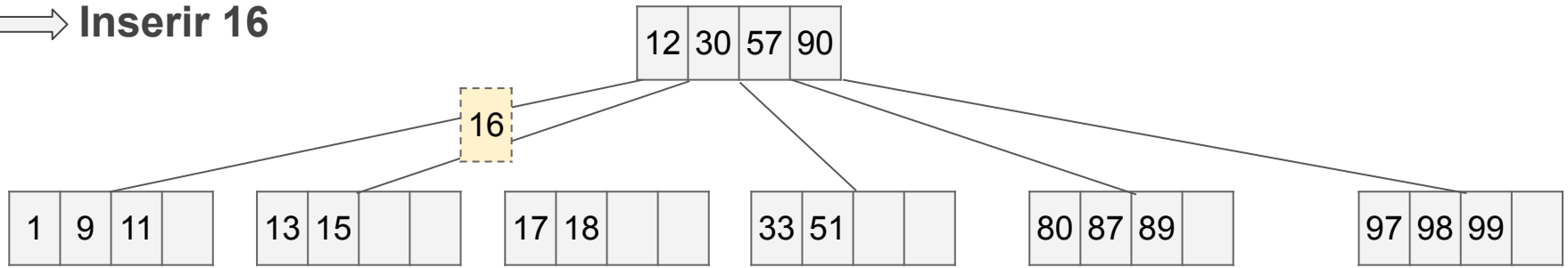
⇨ Inserir 16



`Overflow!`

Exemplo

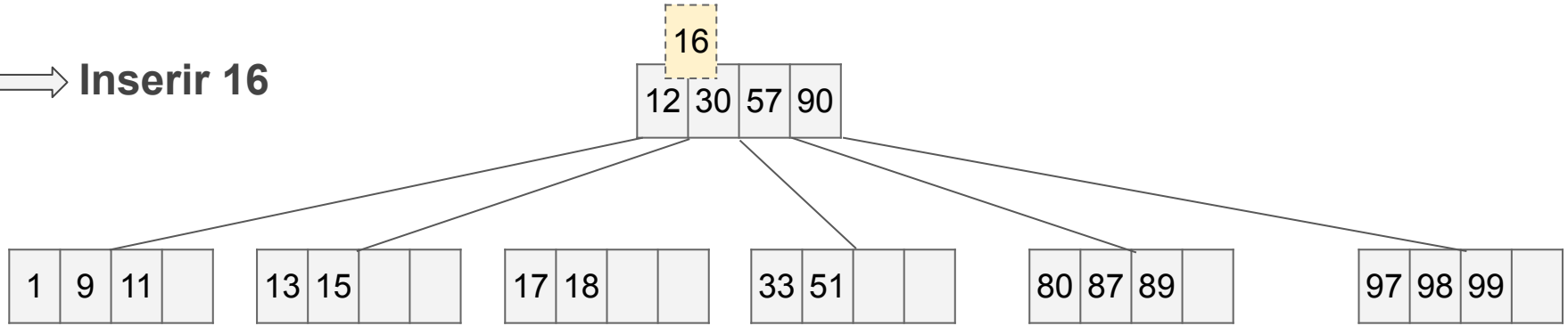
⇨ **Inserir 16**



`Realiza a cisão do nó`

Exemplo

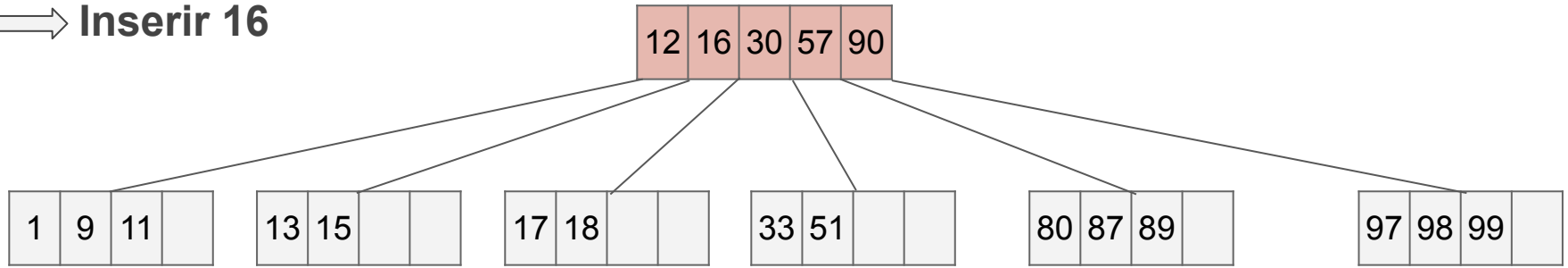
Inserir 16



A chave central é promovida para o pai

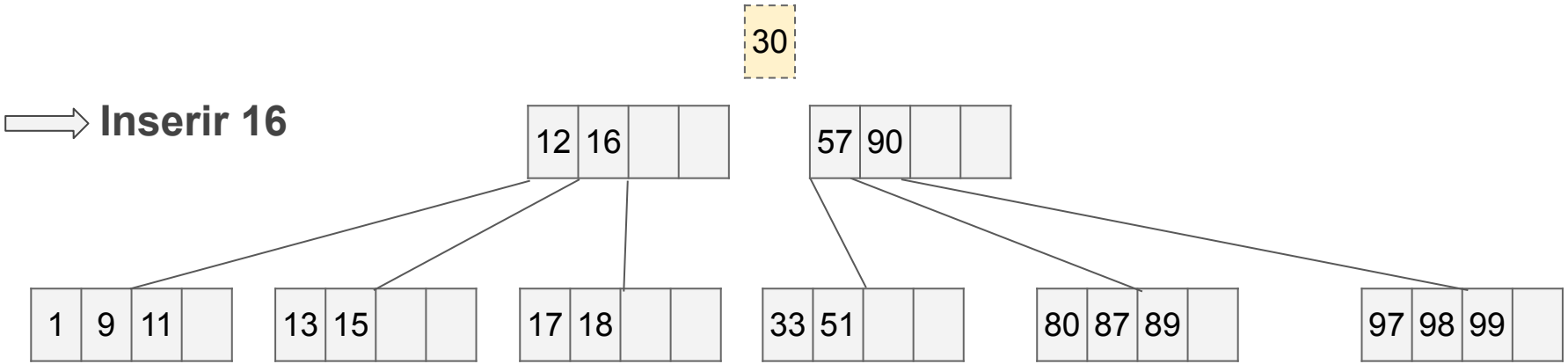
Exemplo

⇨ Inserir 16




```
Overflow!
```

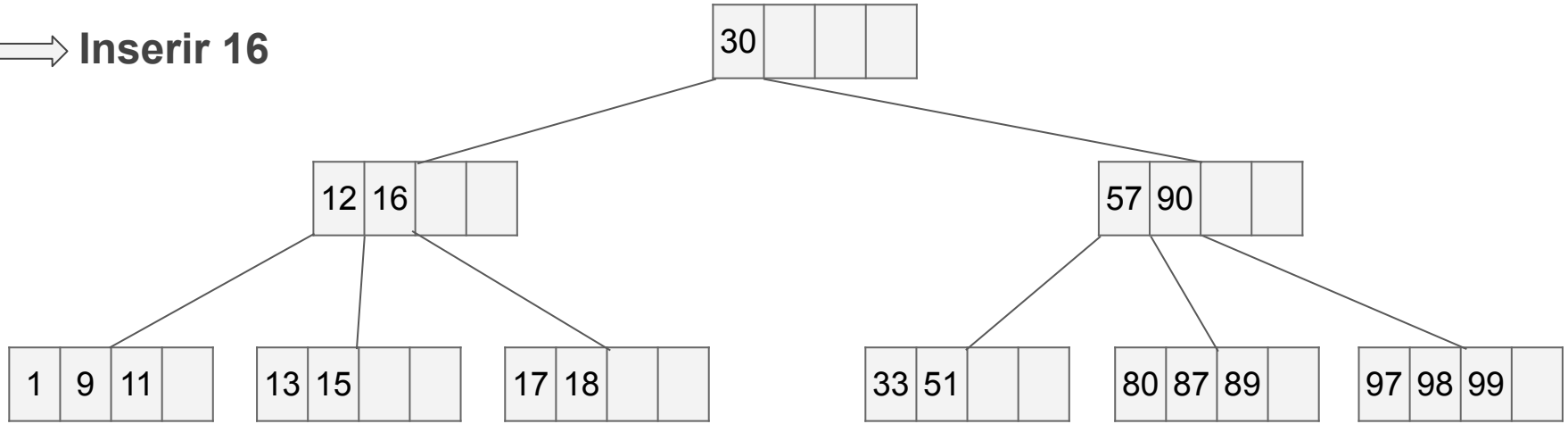
Exemplo



Realiza a cisão da raiz

Exemplo

⇨ Inserir 16



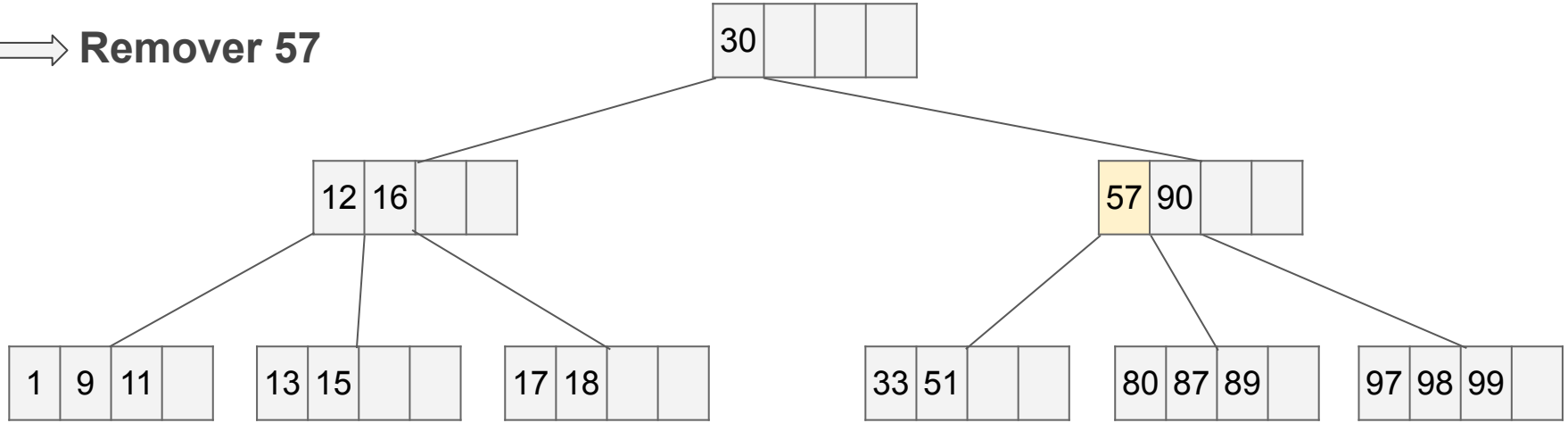
Uma nova raiz é criada

Remoção

- Dois casos possíveis
  - Nó é folha ⇒ deleta a chave
  - Nó não é folha ⇒ similar à ABB, troca a chave com o seu antecessor ou sucessor e deleta a chave na folha
- Pode ocorrer *underflow*
  - Dois procedimentos possíveis
    - **Redistribuição:** chave separadora desce do pai para o nó e uma chave do irmão sobe para o pai
      - Ocorre se o número de chaves do irmão é maior que o mínimo
    - **Junção:** nó deixa de existir e suas chaves são fundidas com as do irmão, mais a chave separadora localizada no nó pai
      - Ocorre se o número de chaves do irmão é exatamente o mínimo

Exemplo

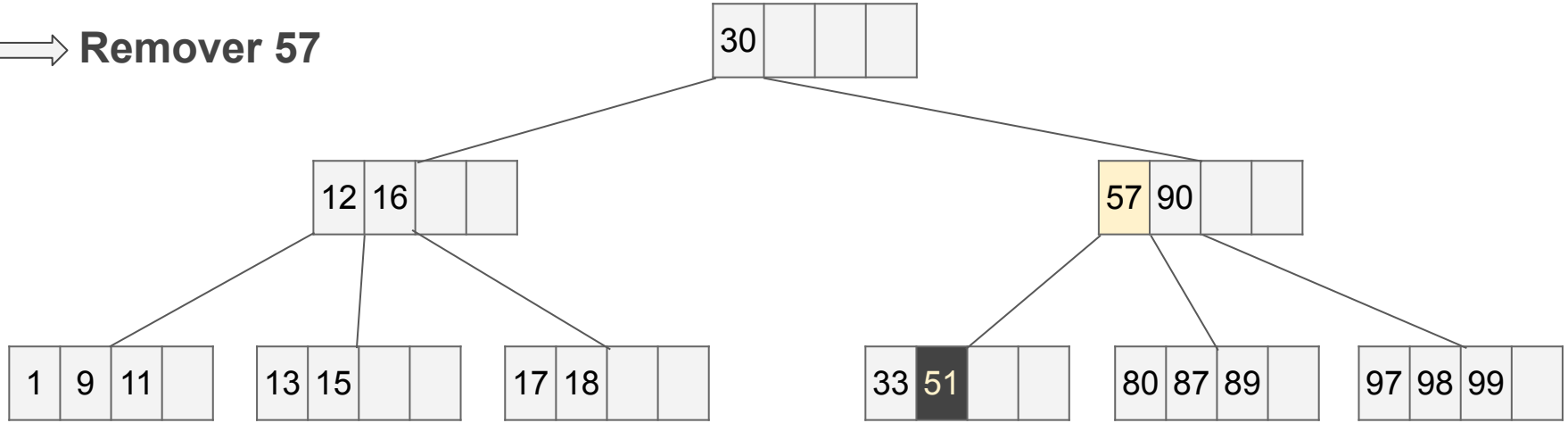
⇨ Remover 57



Chave não se encontra na folha

Exemplo

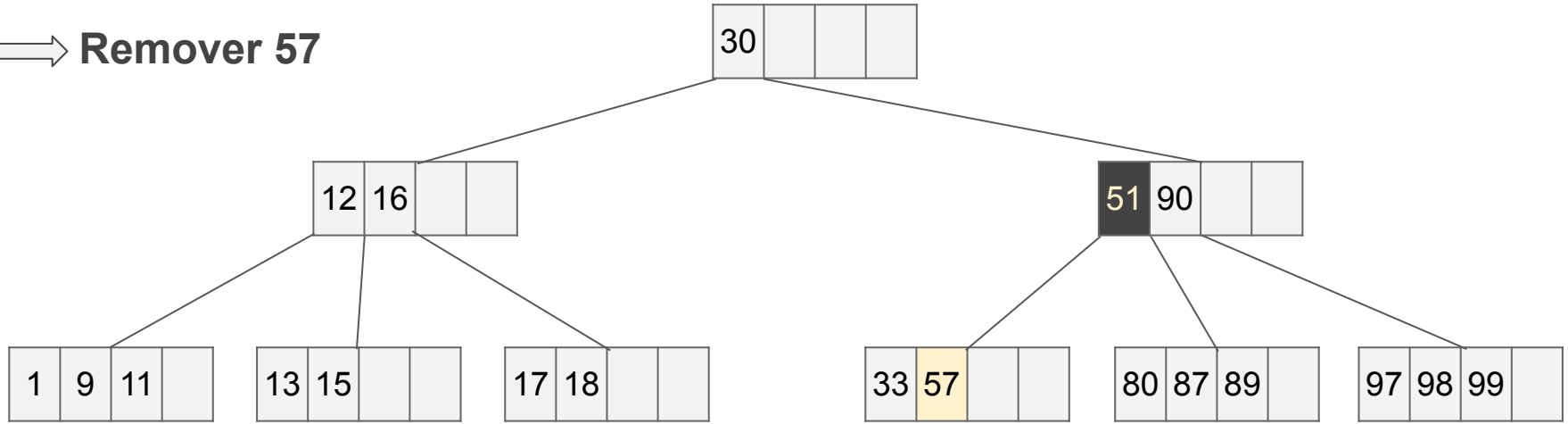
Remover 57



Troca com o antecessor

Exemplo

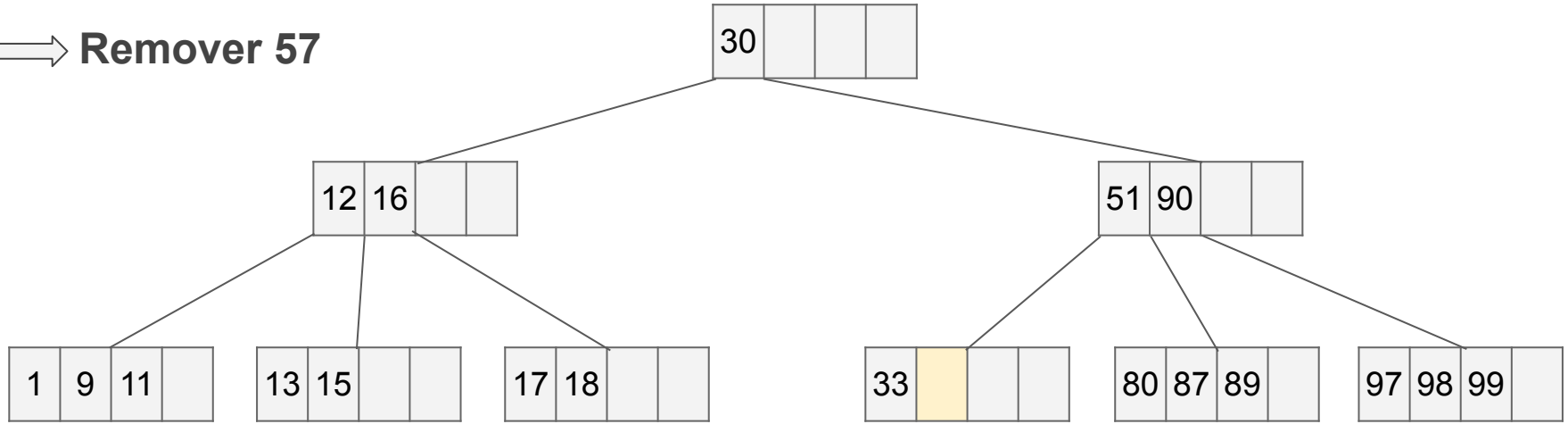
⇨ Remover 57



Troca com o antecessor

Exemplo

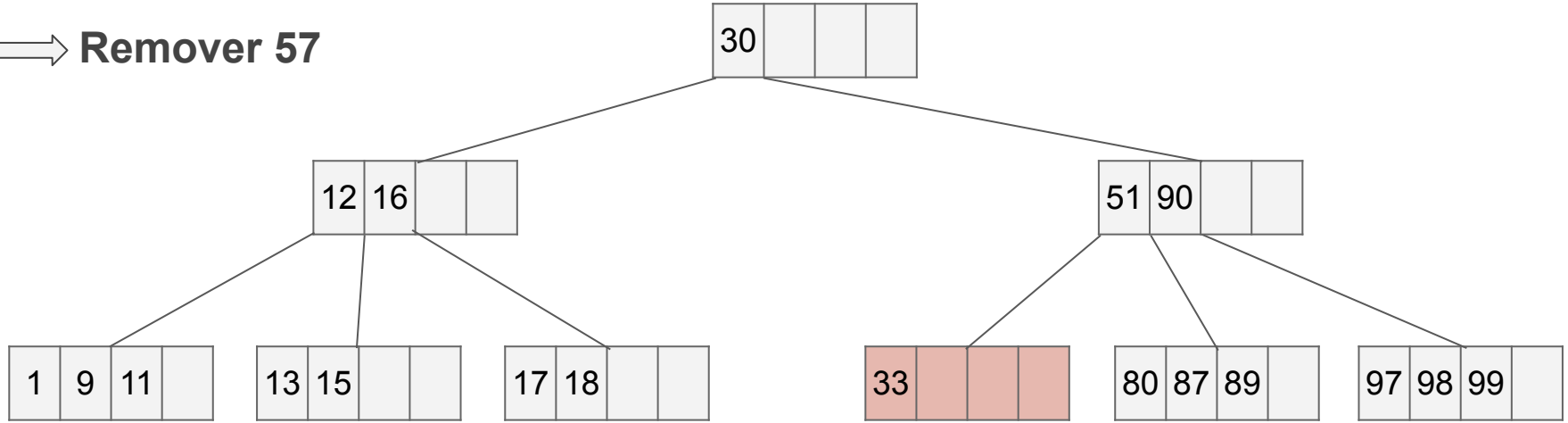
Remover 57



`Remove a chave`

Exemplo

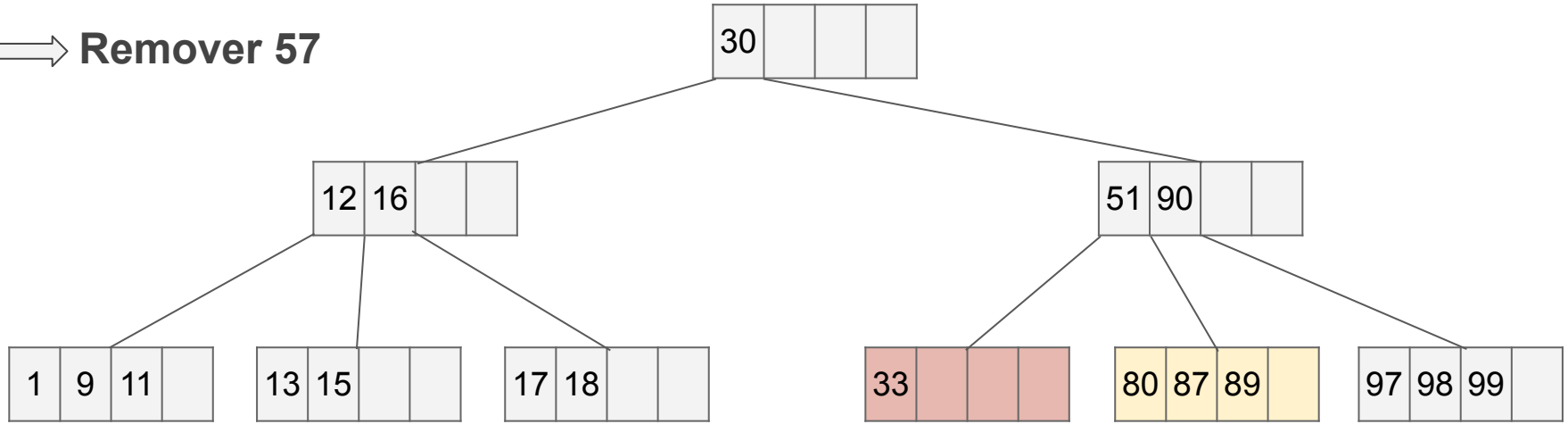
Remover 57



`Underflow!`

Exemplo

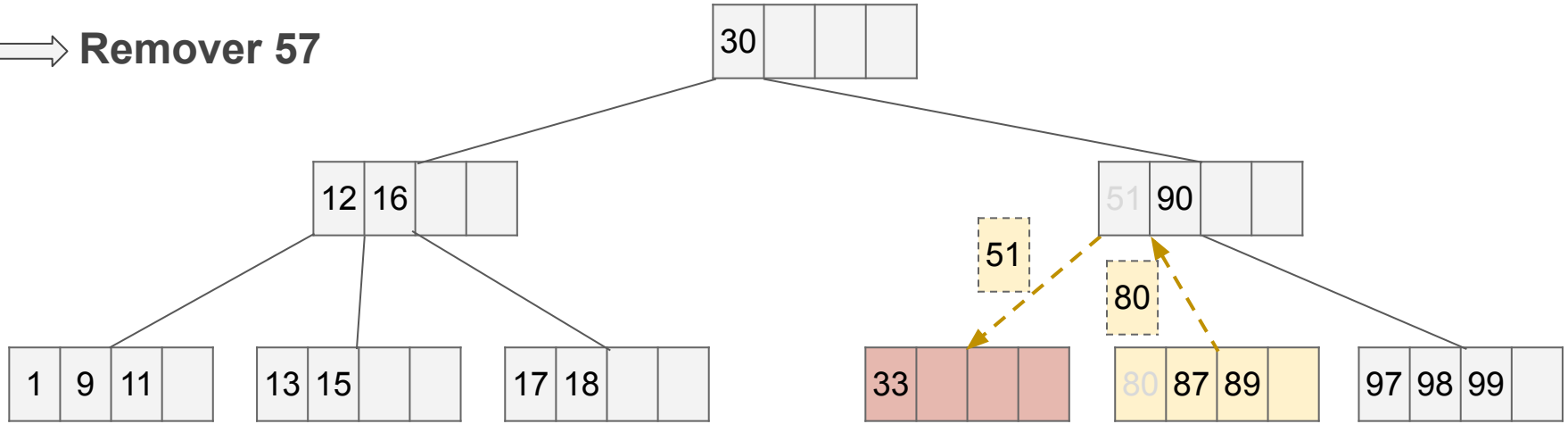
⇨ Remover 57



Irmão possui mais que o mínimo de chaves

Exemplo

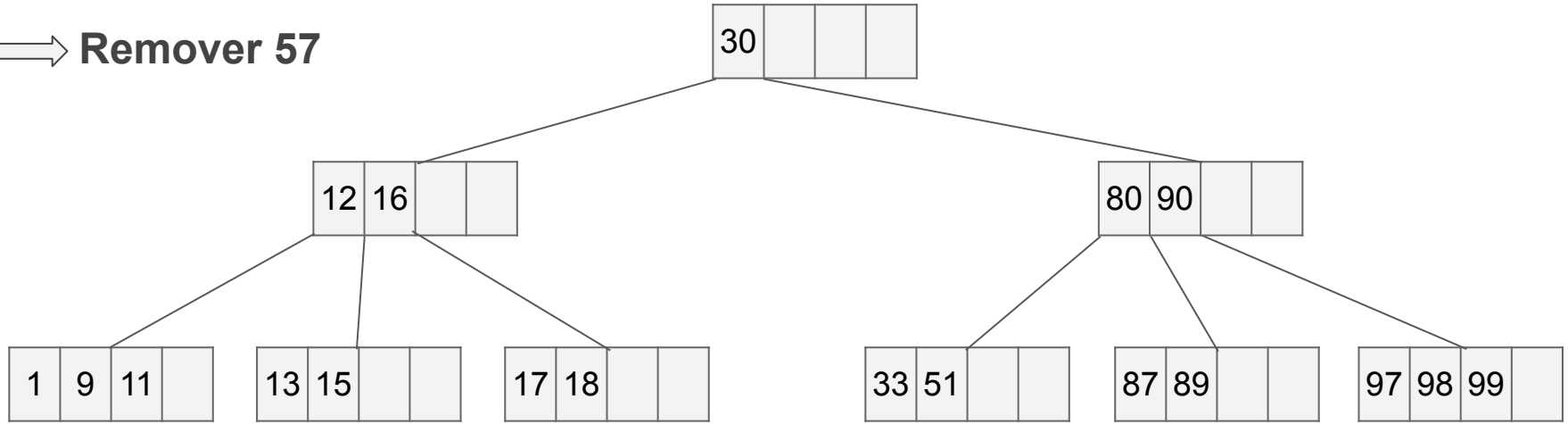
⇨ Remover 57



Realiza a redistribuição

Exemplo

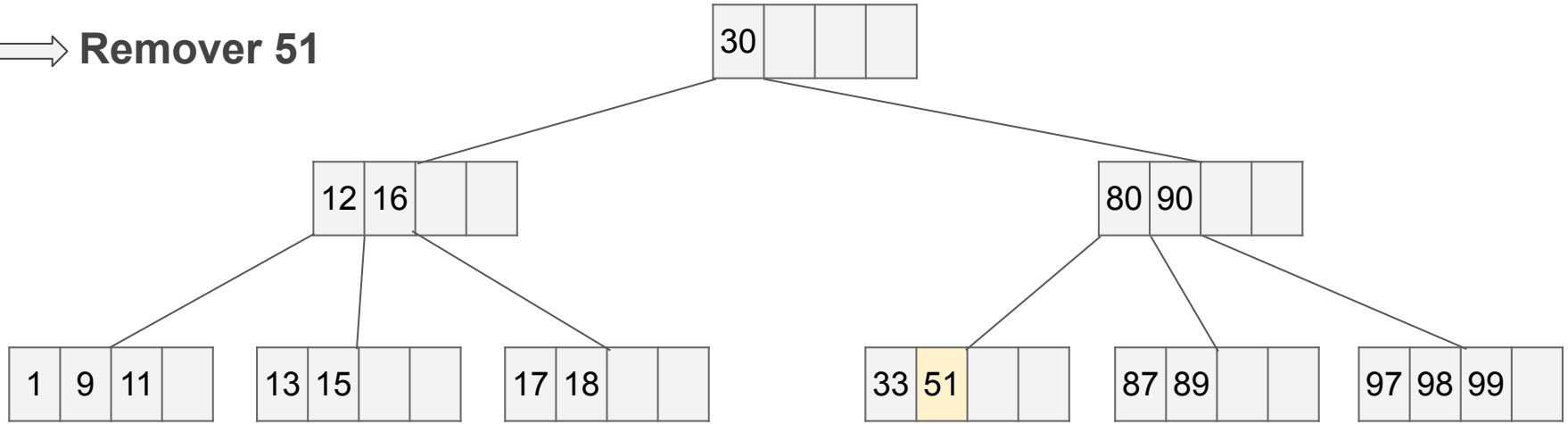
⇨ Remover 57



Realiza a redistribuição

Exemplo

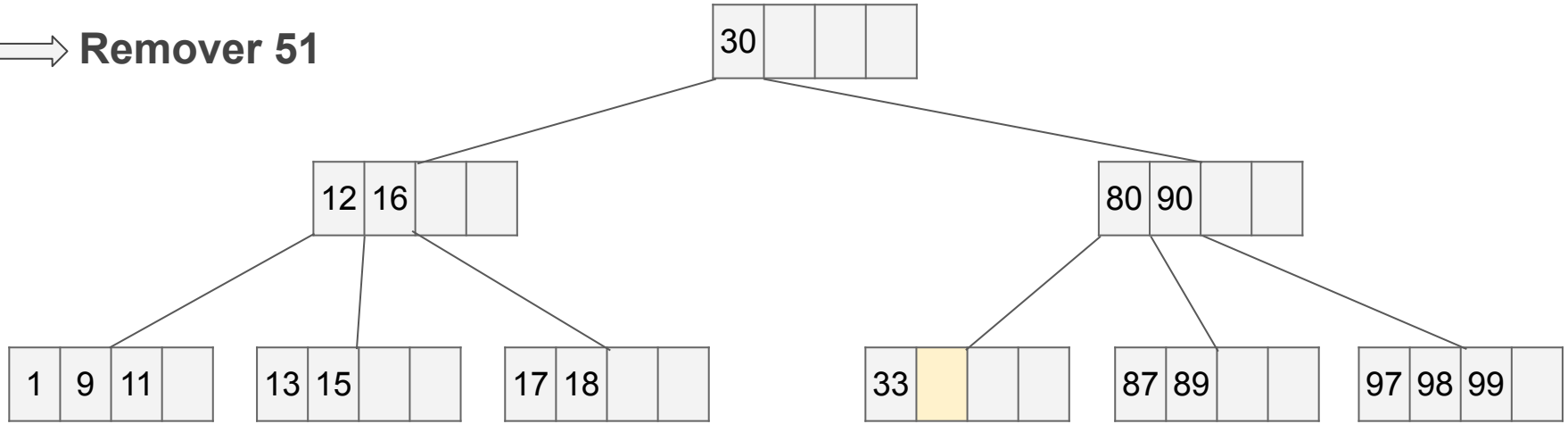
⇨ Remover 51



Chave se encontra na folha

Exemplo

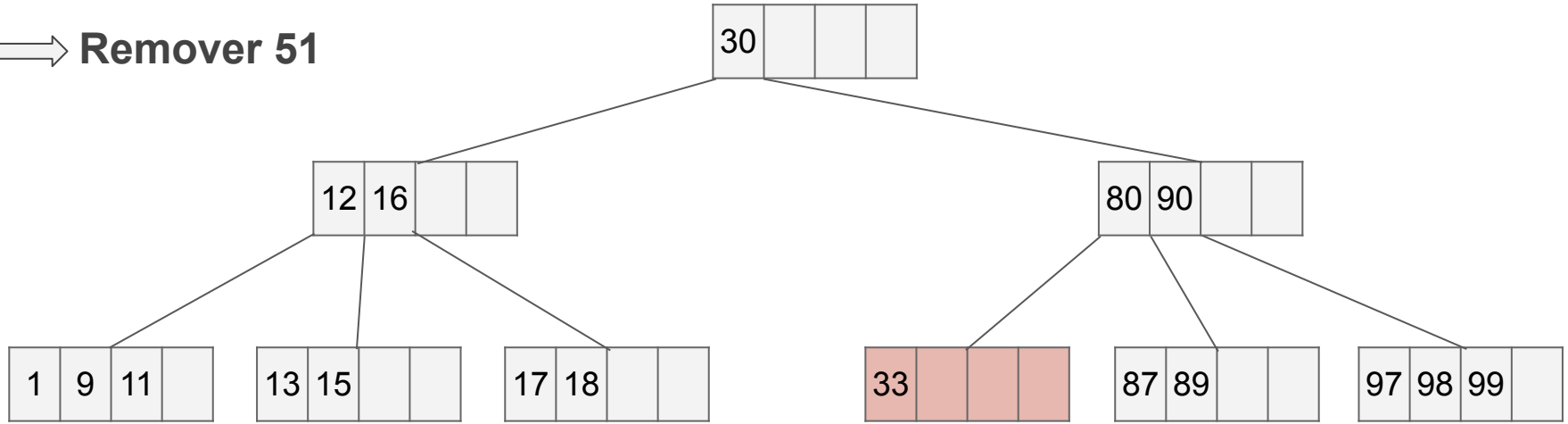
⇨ Remover 51



`Remove a chave`

Exemplo

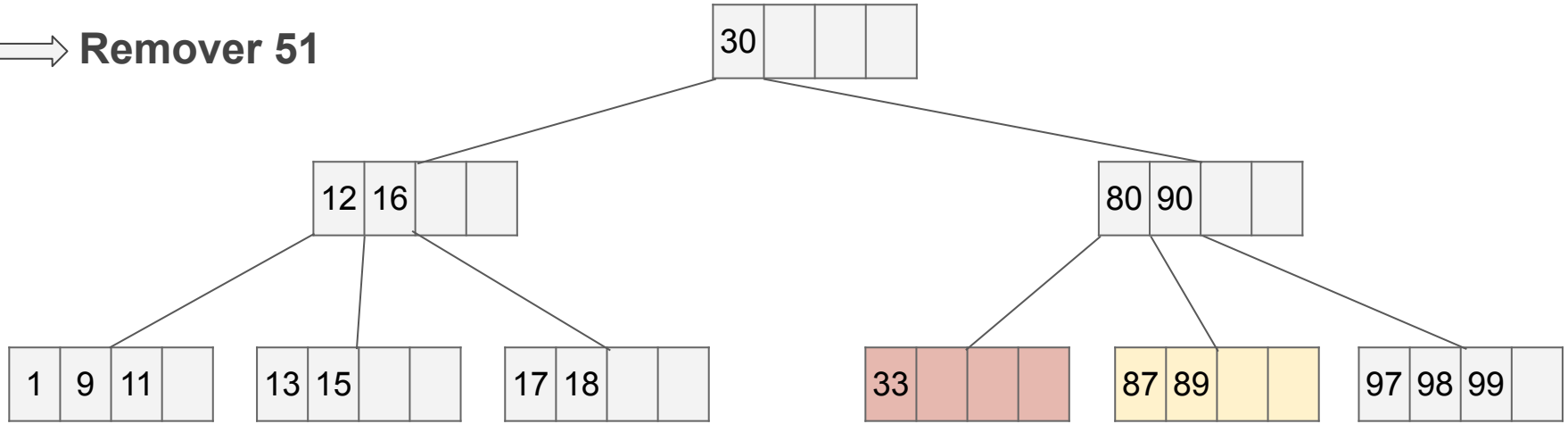
⇨ Remover 51



`Underflow!`

Exemplo

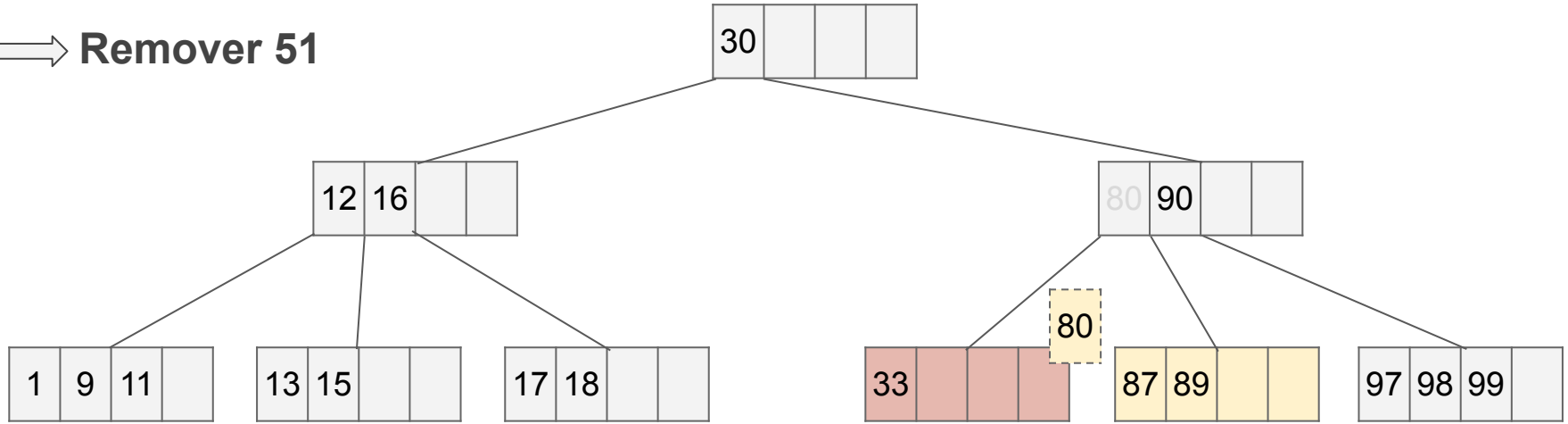
⇨ Remover 51



Irmão tem exatamente o mínimo de chaves

Exemplo

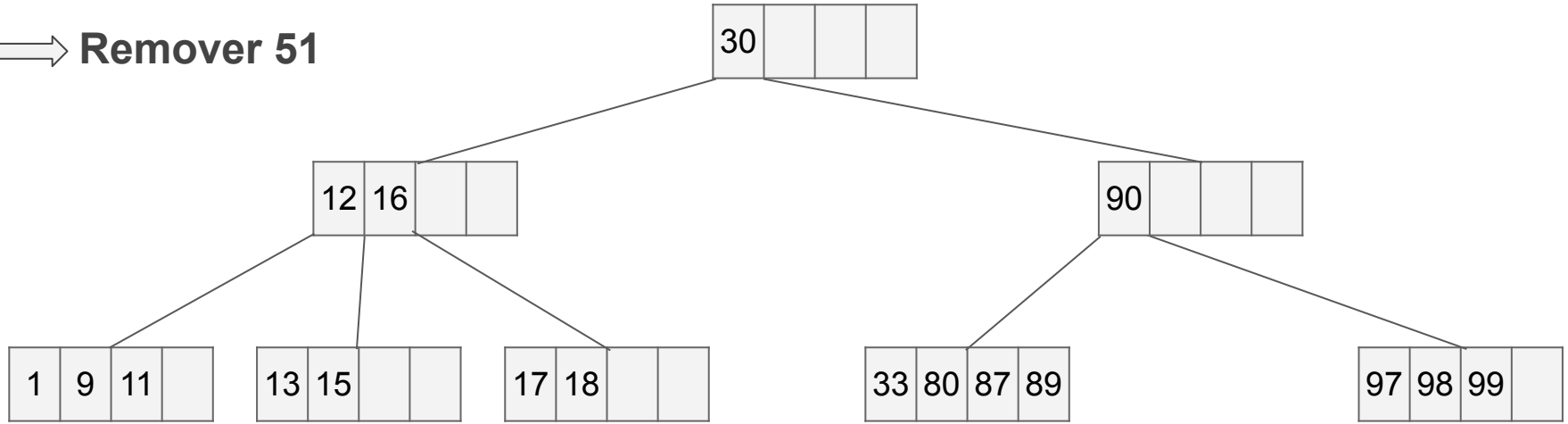
⇨ Remover 51



Realiza a junção

Exemplo

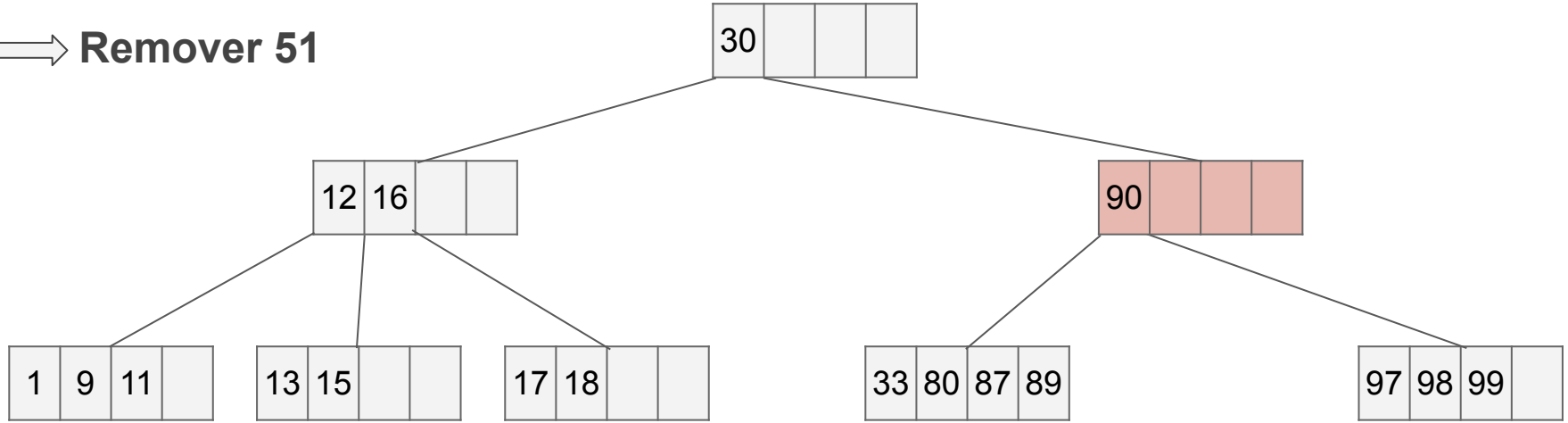
⇨ Remover 51



Realiza a junção

Exemplo

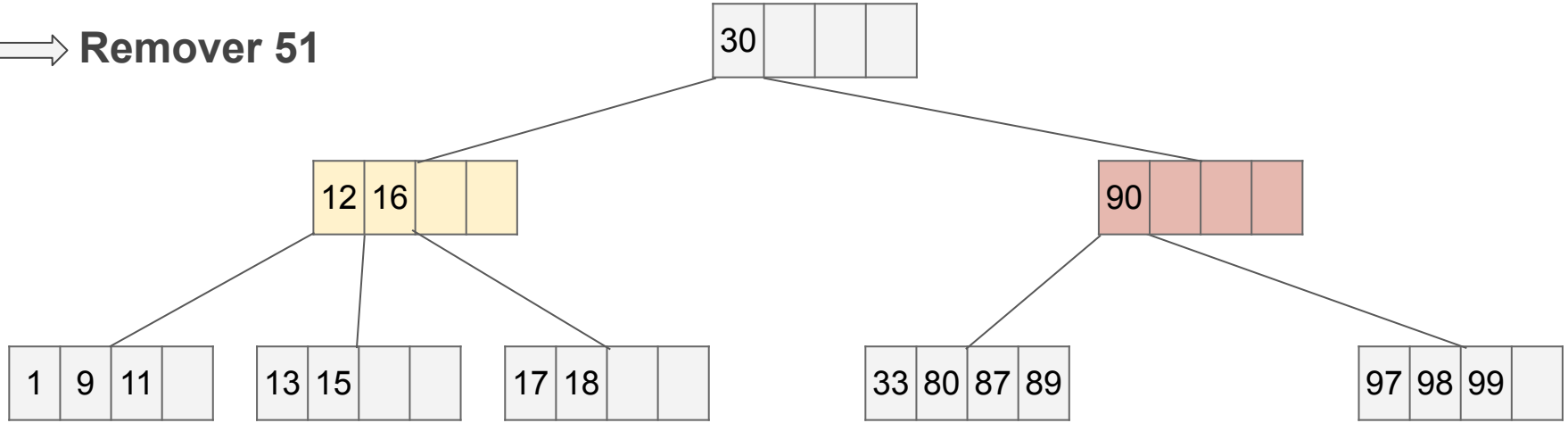
Remover 51



`Underflow!`

Exemplo

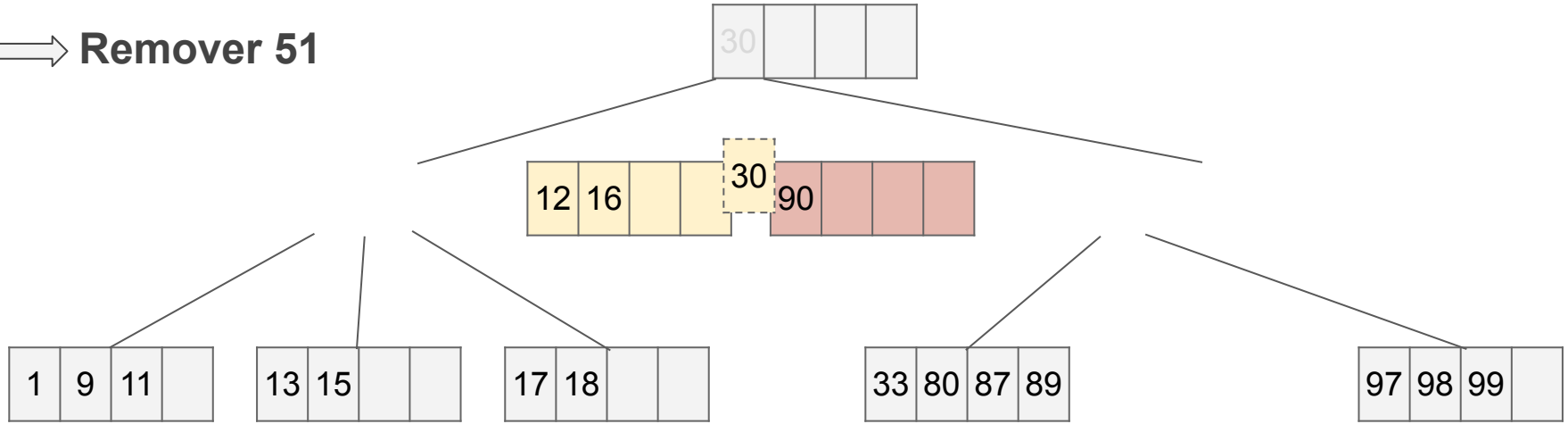
⇨ Remover 51



Irmão tem exatamente o mínimo de chaves

Exemplo

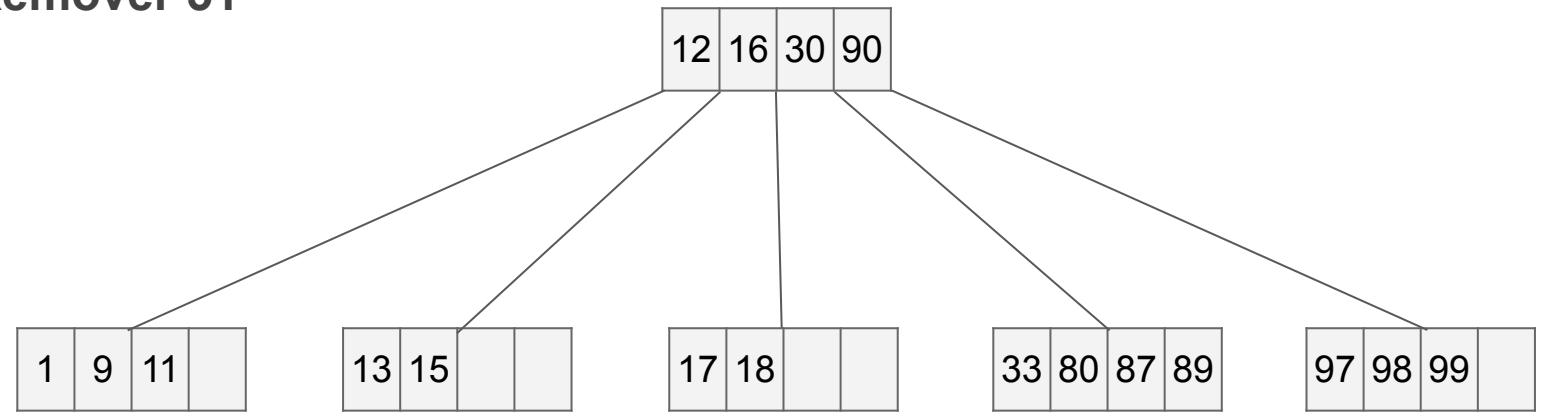
⇨ Remover 51



Realiza a junção

Exemplo

⇨ **Remover 51**



`Realiza a junção`

Análise

- Cada nó visitado representa um acesso ao disco
- Todas as operações (busca, inserção e remoção) são de ordem $O(m \log_m n)$
- Cisão e junção podem propagar até a raiz no pior caso
- Cada nó visitado exige um acesso ao disco
  - Como a altura da árvore é tipicamente pequena, o número de acessos também é pequeno
  - Existem variantes da árvore B que buscam reduzir ainda mais a quantidade de acessos ao disco

Árvore B*

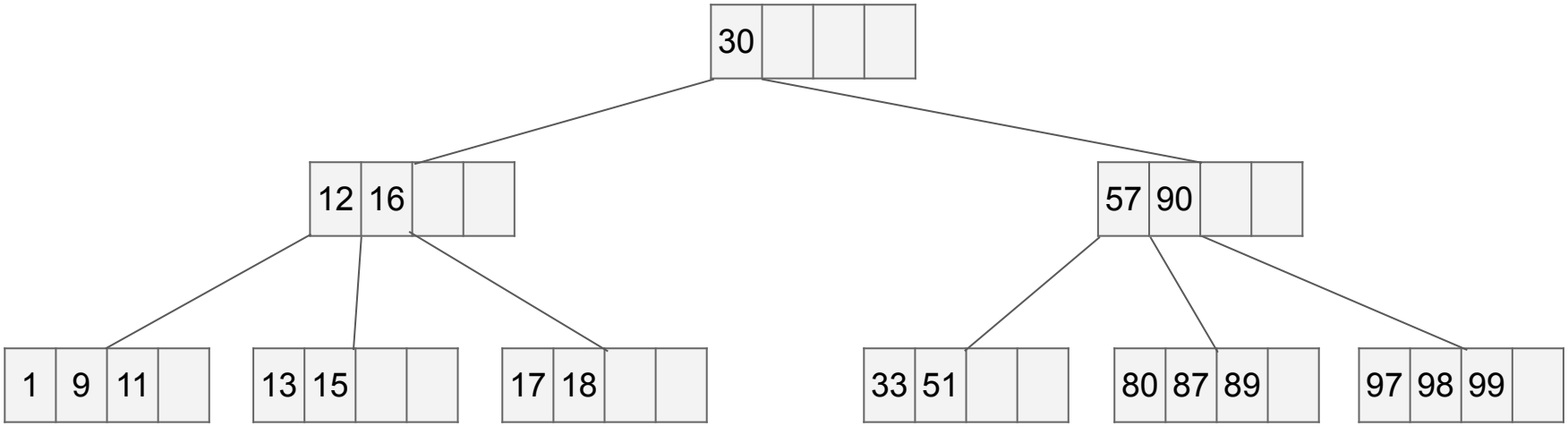
- Cada nó deve ter uma ocupação mínima de ⅔
- Para uma árvore de ordem m, o número de filhos k de um nó é:

$$\left\lfloor \frac{2m-1}{3} \right\rfloor \leq k \leq m-1$$

- Cisão é adiada
  - Só acontece se o irmão estiver cheio, caso contrário redistribui
  - Se ocorrer cisão, as chaves tanto do nó quanto do irmão são divididas em três nós, cada um com ⅔ de ocupação

Árvore B+

- Suponha que queiramos consultar todas as chaves da árvore em ordem ascendente



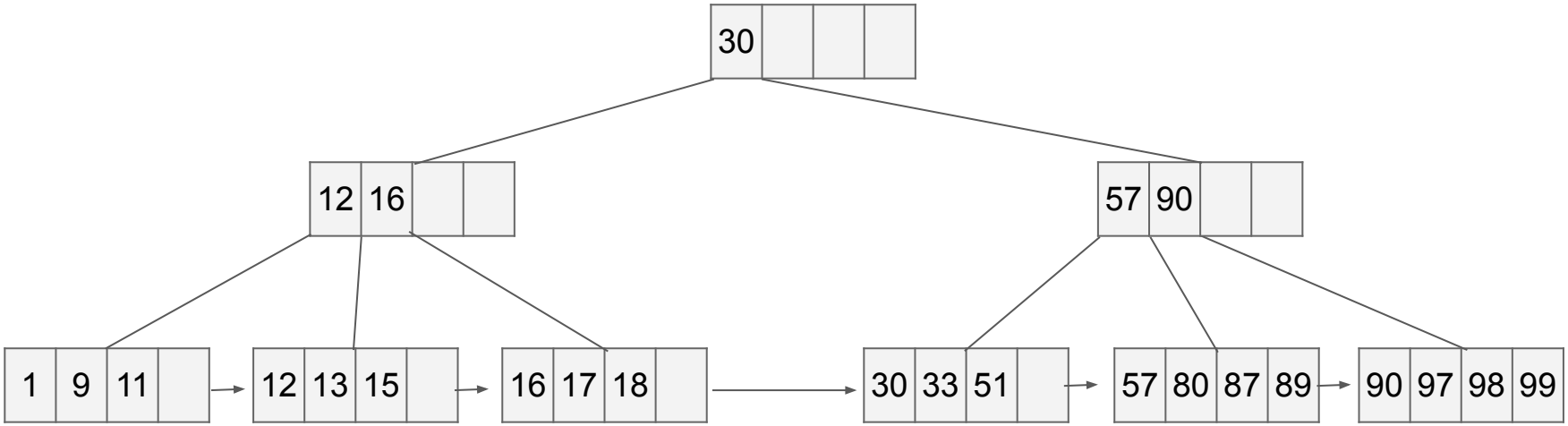
Um percurso in-ordem forçaria um acesso para cada nó

Árvore B+

- Suponha que queiramos consultar todas as chaves da árvore em ordem ascendente
- Solução
  - Manter a referências a dados somente nas folhas
  - Nós internos formam um conjunto de índices
  - As folhas formam um conjunto sequência, com cada nó conectado ao próximo, como em uma lista encadeada
- Aplicações
  - Sistemas de arquivos ⇒ Exs.: FAT, NTFS, XFS, JFS2 e ext4
  - Bancos de dados relacionais ⇒ Exs.: PostgreSQL e MySQL

Árvore B+

➢ Exemplo de uma árvore B+ equivalente à anterior

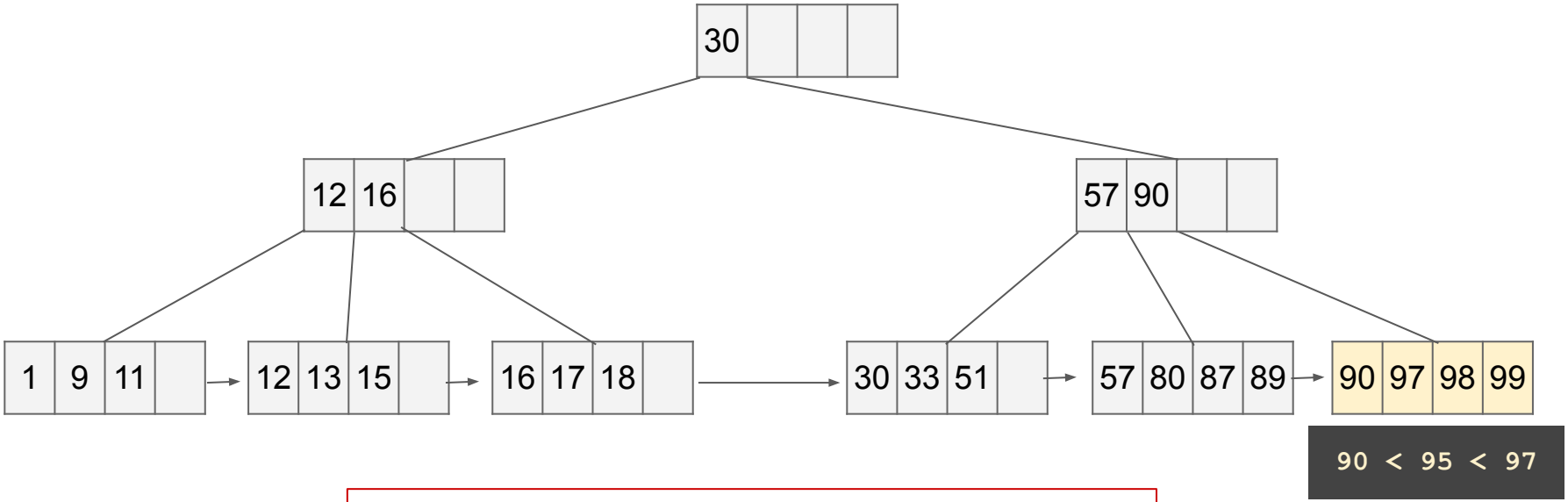


Árvore B+

- Diferenças na inserção
  - Na cisão de uma folha, a chave promovida é **copiada** para o pai, e não movida
  - Cisão em nós internos ocorrem da mesma forma como na árvore B, ou seja, não ocorre cópia
- Diferenças na remoção
  - Se a chave removida tem uma cópia correspondente no índice, o índice não precisa ser removido
    - Caso ocorra underflow em outro momento, a chave separadora no índice é removida, já que não há mais uma chave correspondente na folha

Árvore B+

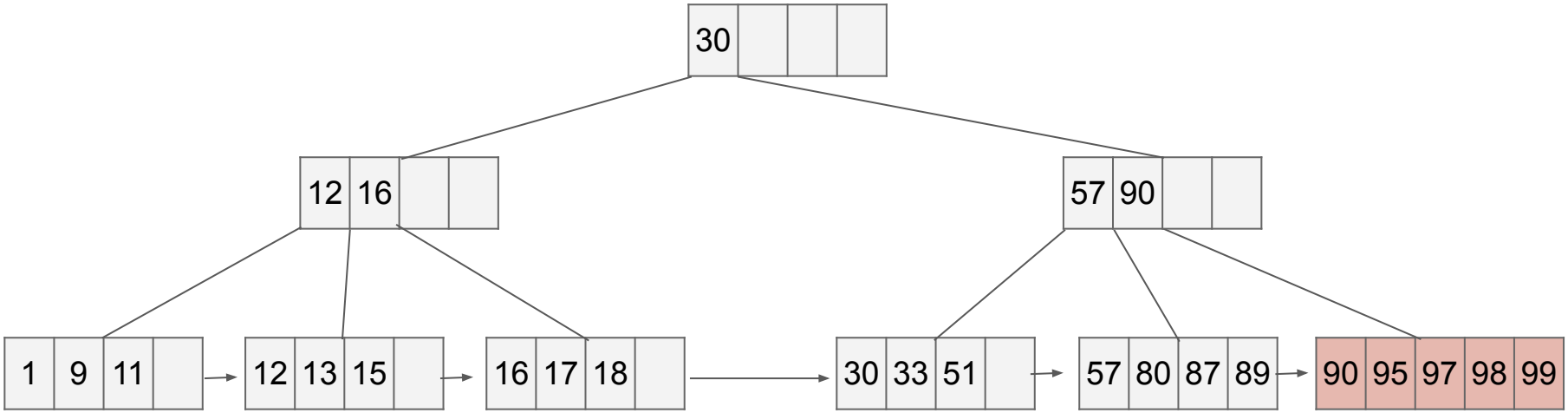
⟹ Inserir 95



Encontra posição de inserção

Árvore B+

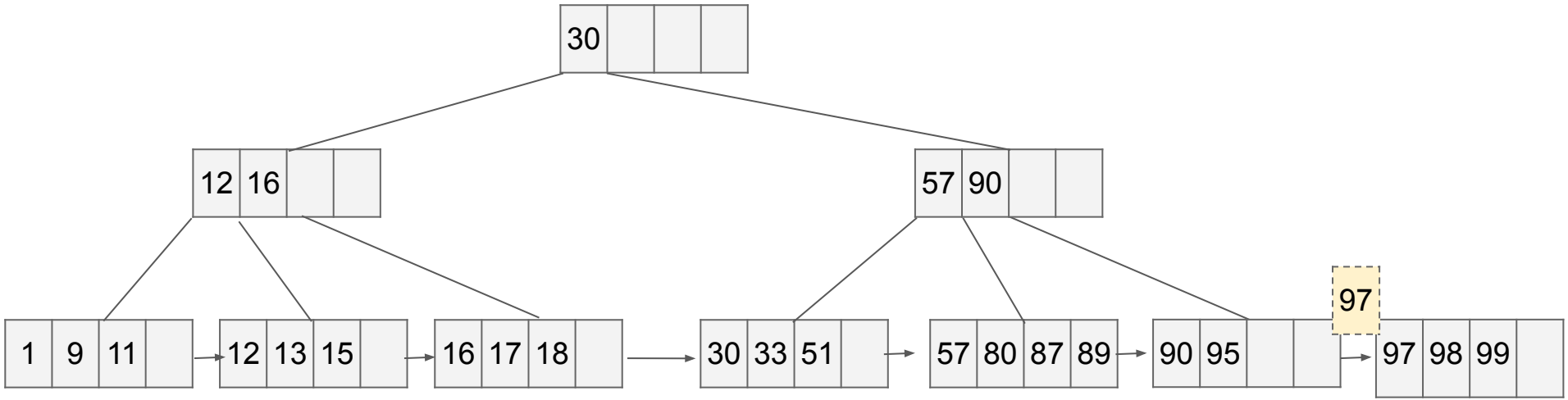
⟹ Inserir 95



`Overflow!`

Árvore B+

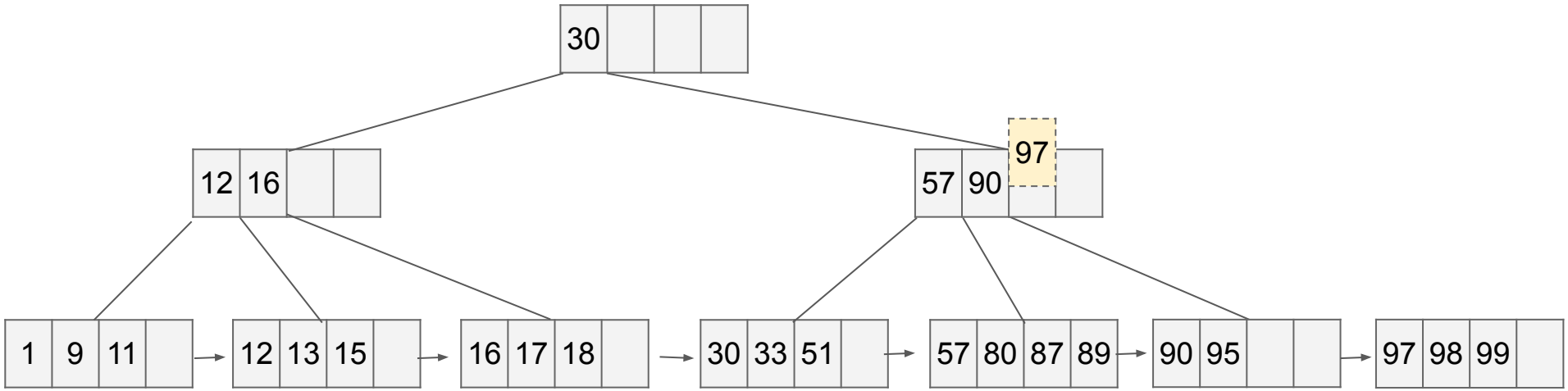
⟹ Inserir 95



Realiza a cisão do nó

Árvore B+

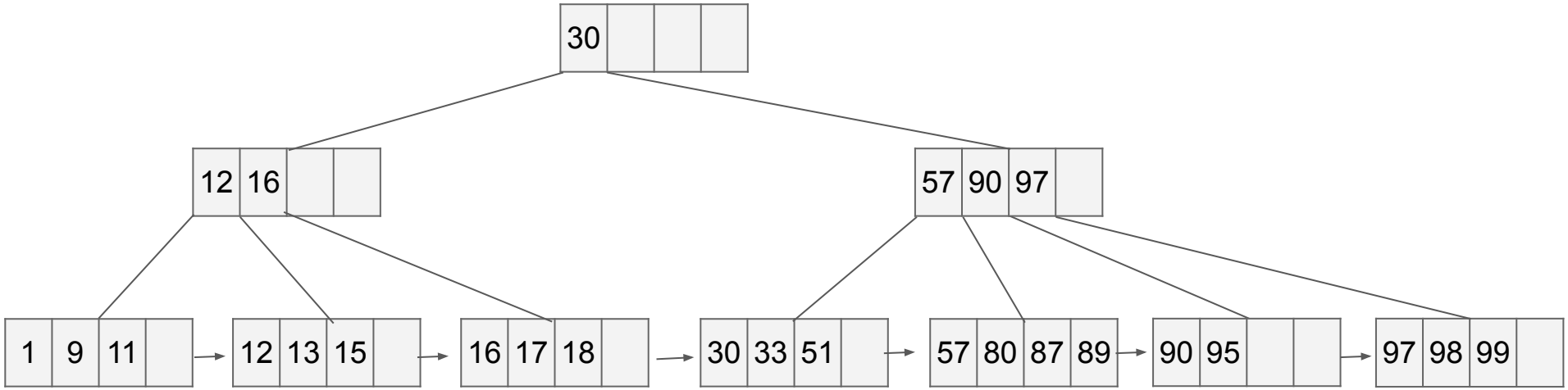
⇨ Inserir 95



A chave central é **copiada** para o pai

Árvore B+

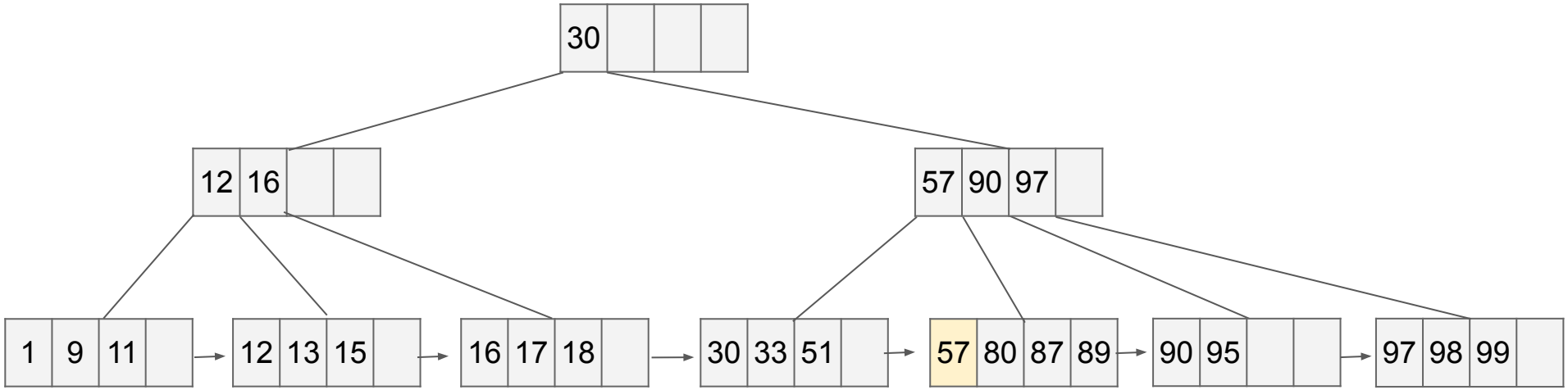
⇨ Inserir 95



A chave central é **copiada** para o pai

Árvore B+

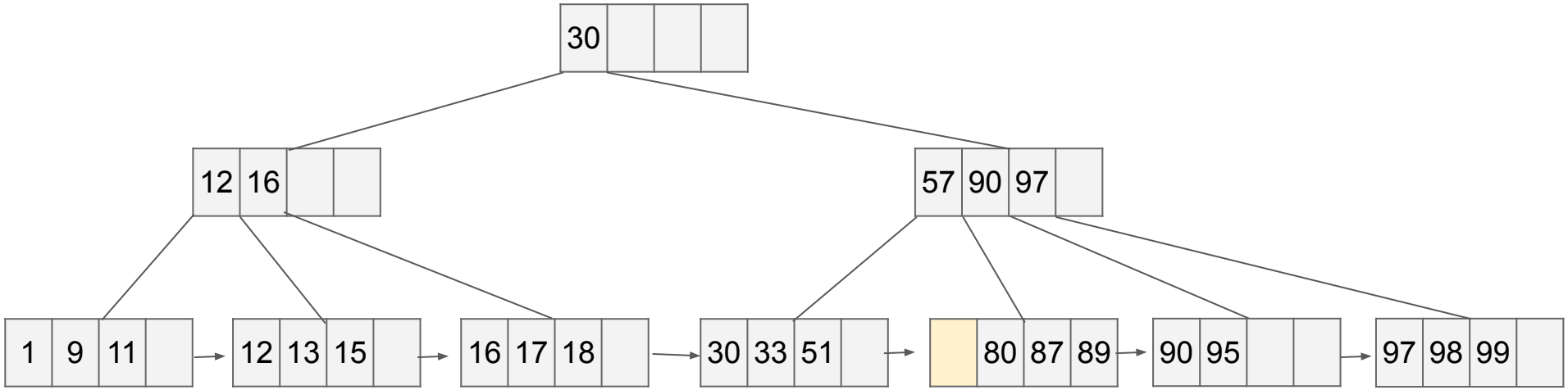
⟹ Remover 57



Encontra a chave a ser removida

Árvore B+

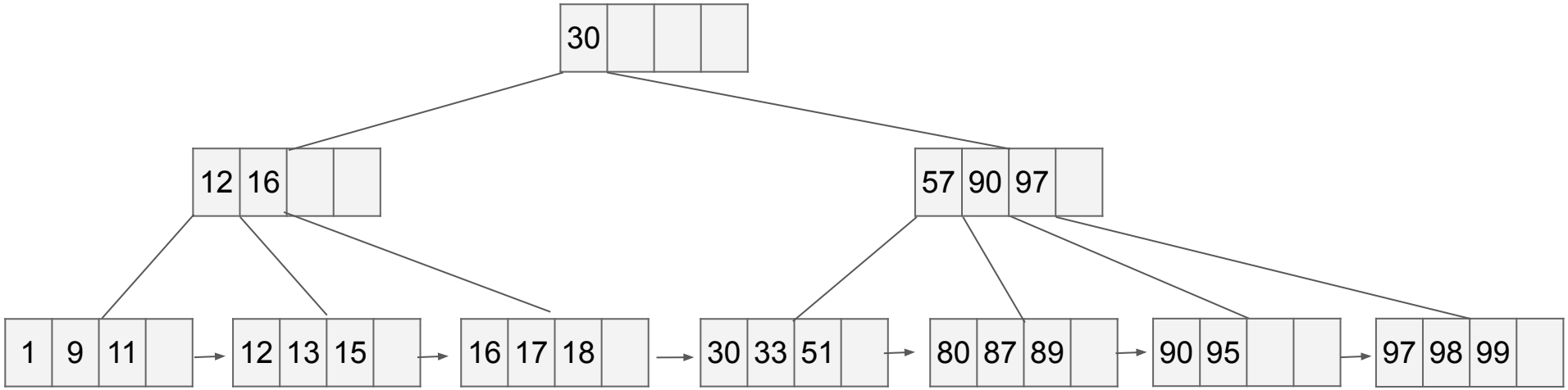
⇨ Remover 57



Remove a chave

Árvore B+

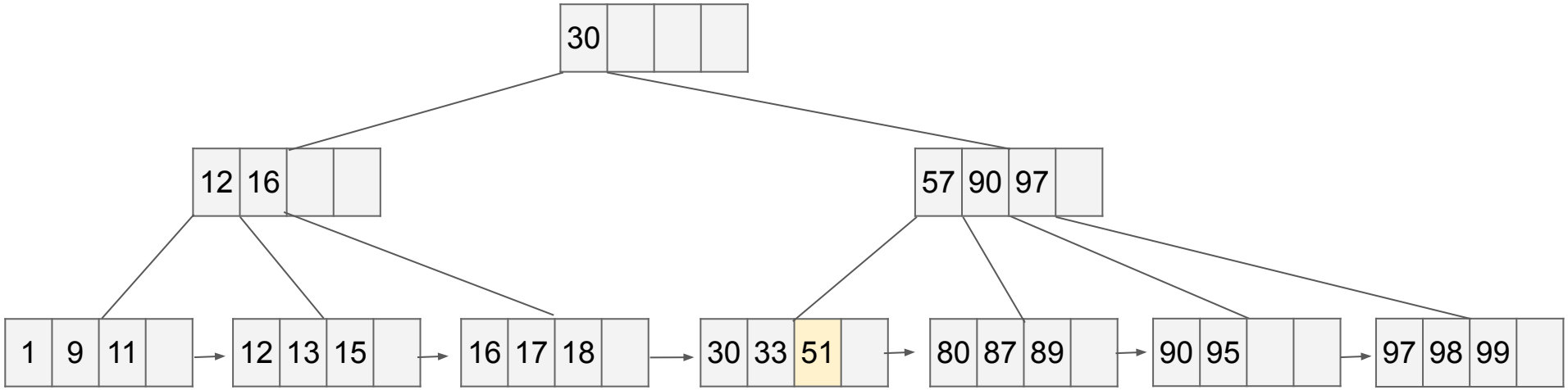
⇨ Remover 57



Note que o índice permanece como separador

Árvore B+

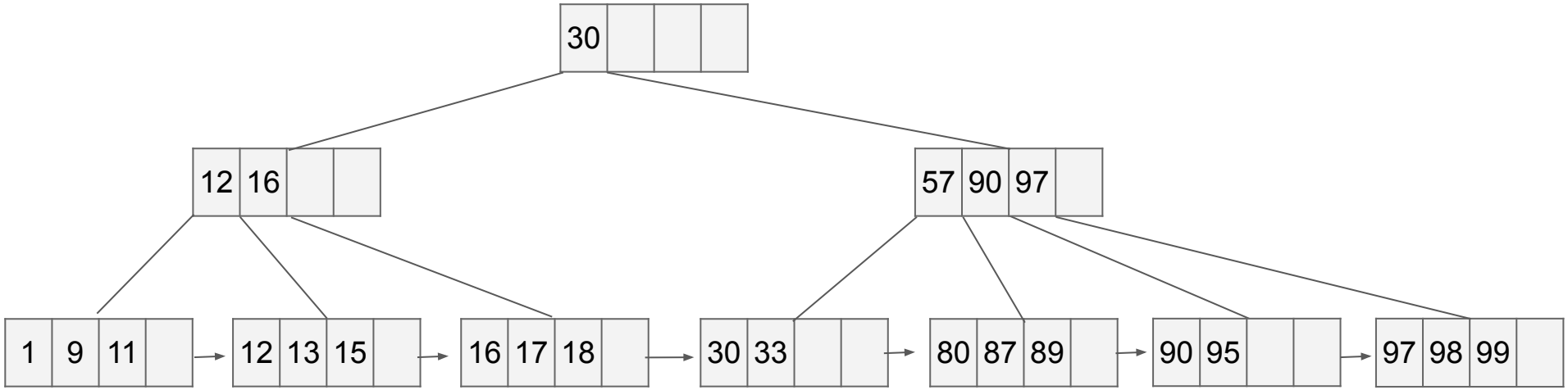
⇨ Remover 51



Encontra a chave a ser removida

Árvore B+

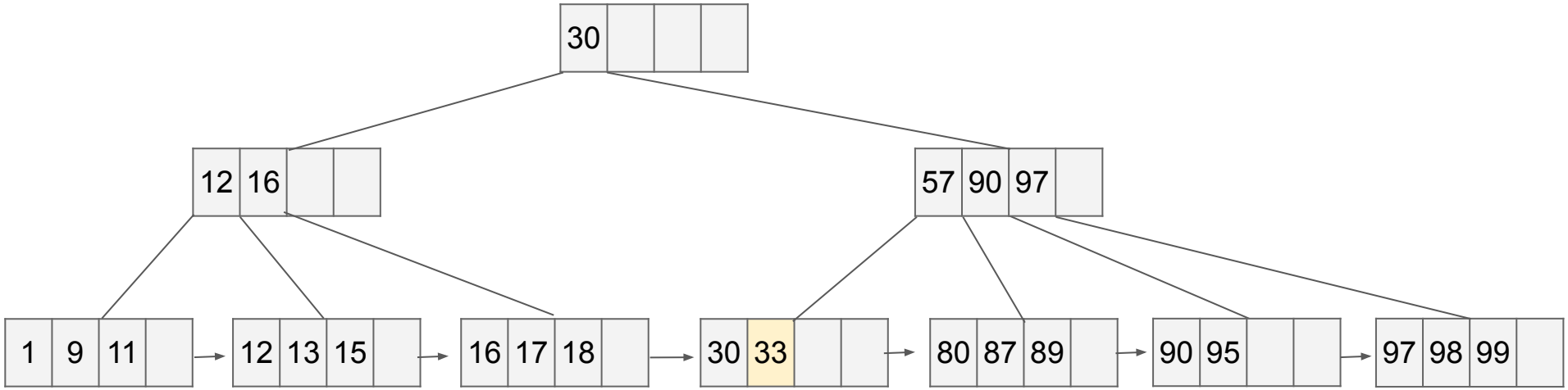
⇨ Remover 51




```
Remove a chave
```

Árvore B+

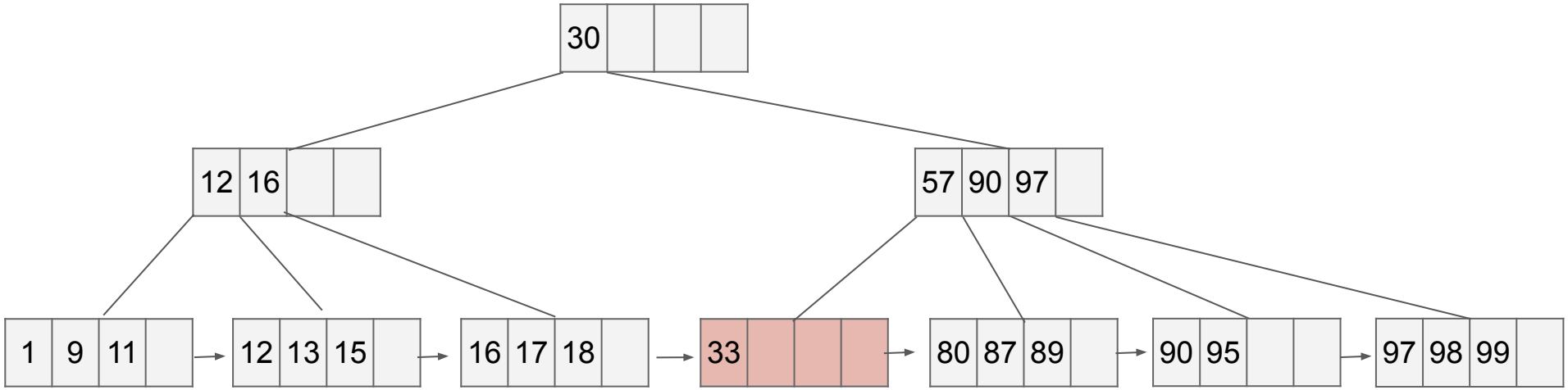
⟹ **Remover 33**



`Encontra a chave a ser removida`

Árvore B+

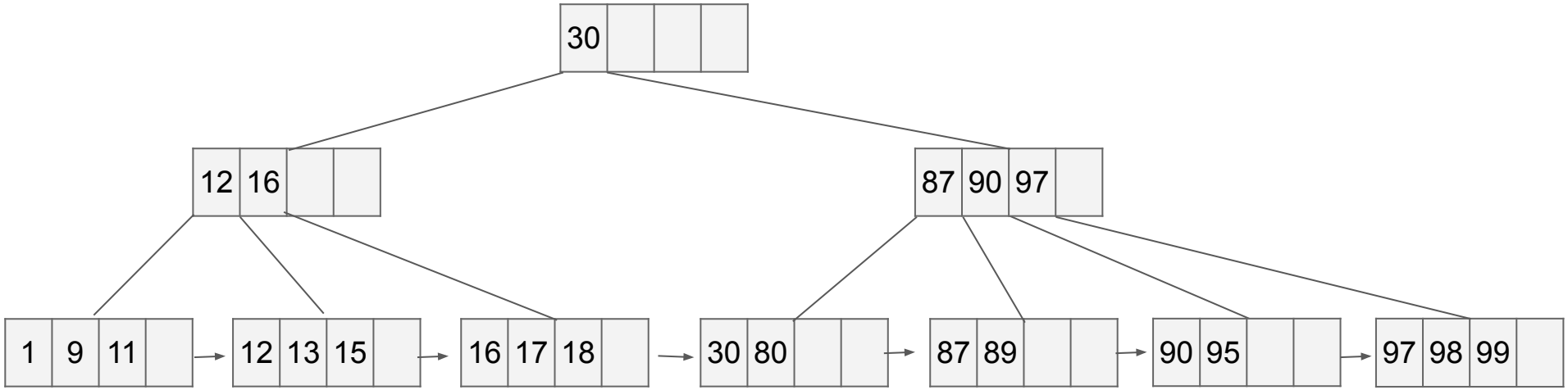
⇨ Remover 33



`Underflow!`

Árvore B+

⇨ **Remover 33**



`Realiza a redistribuição`

Exercício

Considerando uma árvore B de ordem 5, faça a inserção das seguintes chaves:

8,14,2,15,3,1,16,6,5,27,37,18,25,7,13,20,22,23,24

Referências

- DROZDEK, Adam. Data Structures and Algorithms in C++, Fourth Edition, cap. 7. Cengage Learning, 2013.
- CORMEN, T.; Leiserson, C.; Rivest, R; Stein, C. Introduction to Algorithms, Third Edition, cap. 18. MIT Press, 2009.
- SOUZA, Jairo F. Notas de aula de Estrutura de Dados II. 2016. Disponível em: http://www.ufjf.br/jairo_souza/ensino/material/ed2/